Exercise

- **Sort**
- Write a program that sorts an array in descending order.

Range-based for loop for arrays

Range-based for loop for arrays

- The range-based for loop is a loop that iterates once for each element in an array
- Each time the loop iterates, it copies an element from the array to a variable
- The range-based for loop was introduced in C++ 11
- For example, if you use the range-based for loop with an eight-element array, the loop will iterate eight times.
- The range-based for loop automatically knows the number of elements in an array
- No need of a counter variable to control its iterations, as with a regular for loop
- Additionally, you do not have to worry about stepping outside the bounds of an array when you use the range-based for loop.

General format of the range-based for loop

- General format:

  for *(dataType rangeVariable : array)*
  statement;

- *dataType* is the data type of the range variable. It must be the same as the data type of the array elements, or a type that the elements can automatically be converted to.
- *rangeVariable* is the name of the range variable.
  - This variable will receive the value of a different array element during each loop iteration.
  - During the first loop iteration, it receives the value of the first element.
  - During the second iteration, it receives the value of the second element, and so forth.
- *array* is the name of an array on which you wish the loop to operate.
  - The loop will iterate once for every element in the array.
- *statement* is a statement that executes during a loop iteration. If you need to execute more than one statement in the loop, enclose the statements in a set of braces.

Example

- Assume that you have the following array definition:

```
int numbers[] = { 3, 6, 9 };
```

- The following range-based for loop to display the contents of the numbers array:

```
for (int val : numbers)
cout << val;
```

- The numbers array has three elements, this loop will iterate three times
  - The first time it iterates, the val variable will receive the value in numbers[0]
  - During the second iteration, val will receive the value in numbers[1]
  - During the third iteration, val will receive the value in numbers[2]
  - The code's output will be: 369

Example: Program

```
1   // This program demonstrates the range-based for loop.
2   #include <iostream>
3   using namespace std;
4
5   int main()
6   {
7       // Define an array of integers.
8       int numbers[] = { 10, 20, 30, 40, 50 };
9
10      // Display the values in the array.
11      for (int val : numbers)
12          cout << val << endl;
13
14      return 0;
15  }
```

Program Output

```
10
20
30
40
50
```

Example program2

```
1  // This program demonstrates the range-based for loop.
2  #include <iostream>
3  #include <string>
4  using namespace std;
5
6  int main()
7  {
8      string planets[] = { "Mercury", "Venus", "Earth", "Mars",
9                           "Jupiter", "Saturn", "Uranus",
10                          "Neptune", "Pluto (a dwarf planet)" };
11
12     cout << "Here are the planets:\n";
13
14     // Display the values in the array.
15     for (string val : planets)
16         cout << val << endl;
17
18     return 0;
19 }
```

Program Output

```
Here are the planets:
Mercury
Venus
Earth
Mars
Jupiter
Saturn
Uranus
Neptune
Pluto (a dwarf planet)
```

Range-Based for Loop vs Regular for Loop

- The range-based for loop can be used in any situation where you need to step through the elements of an array, and you do not need to use the element subscripts.
- It will not work, however, in situations where you need the element subscript for some purpose.
- In those situations, you need to use the regular for loop.

The char array

The char array (Example)

```
#include<iostream>
using namespace std;
int main(){
        char array[4]={'a','b','c', '\0'};
        //alternate way where compiler automatically add '\0' at the end
        //char array[]={"abc"};
        cout<<array;
        return 0;
}
```

Output:

abc

The string array (implementation)

- Using Two-dimensional Character Arrays:
  - This representation uses the two-dimensional arrays where each element is the intersection of a row and column number and represents a string
  - String arrays or an array of strings can be represented using a special form of two-dimensional arrays.
- In this representation, we use a two-dimensional array of type characters to represent a string.
  - The first dimension specifies the number of elements i.e. strings in that array and the second dimension specifies the maximum length of each element in the array.

Declaration of string array

- So we can use a general representation as shown below.

char “stringarrayname” [“number of strings”] [“maximum length of the string”];

- Example:

char string_array[10] [20];

  - The above declaration declares an array of strings named ‘string_array’ which has 10 elements and the length of each element is not more than 20.

- We can declare and initialize an array of animals using strings in the following way:

char animals [5] [10] = {“Lion”, “Tiger”, “Deer”, “Cat”, “Kangaroo”};

Example

```
#include <iostream>
using namespace std;
int main()
{
    char strArray[5] [6] = {"one", "two", "three", "four", "five"};
    cout<<"String array is as follows:"<<endl;
    for(int i=0;i<5;i++)
    {
        cout<<"Element "<<i<<"= "<<strArray[i]<<endl;
    }

  return 0;
}
```

Output:

String array is as follows:
Element 0= one
Element 1= two
Element 2= three
Element 3= four
Element 4= five

Advantages and disadvantages

- Advantage:
  - Easy access to elements
  - Simple to program.
- Disadvantage:
  - Both the dimensions of array i.e. number of elements and the maximum length of the element are fixed and cannot be changed
  - If the string length is specified as 100, and we have all the elements that are lesser in length, then the memory is wasted.

Using string Keyword

- Use the string keyword of C++ to declare and define string arrays.
  - Unlike character arrays, here we have only 1D array.
  - The sole dimension specifies the number of strings in the array.
- The general syntax

  string “array name” [“number of strings”];
  - Note that we do not specify the maximum length of string here.
  - This means that there is no limit on the length of the array elements.
- Example, we can declare an array of color names in the following way.

  string colors[5];
- We can further initialize this array as shown below:

  string colors[5] = {“Red”, “Green”, “Blue”, “Orange”, “Brown”};

The string keyword and char array

- Advantages;
  - The output of the program is the same but the way it is achieved is different as we define an array of strings using the string keyword
  - The array of strings have no limit on the length of the strings in the array
  - Since there is no limit, we do not waste memory space as well
- Disadvantages:
  - On the downside, this array has a fixed size. We need to declare the size of the array beforehand.

Example

```
#include <iostream>
using namespace std;
int main()
{
    string numArray[5] = {"one", "two", "three", "four", "five"};
    cout<<"String array is as follows:"<<endl;
    for(int i=0;i<5;i++)
    {
        cout<<"Element "<<i<<"= "<<numArray[i]<<endl;
    }

  return 0;
}
```

Output:

String array is as follows:
Element 0= one
Element 1= two
Element 2= three
Element 3= four
Element 4= five

Looping Through a char array

```
#include <iostream>
using namespace std;
int main() {
char word[] = "HelloWorld";
    int i = 0;
    cout << "Characters in the char array:" << endl;

while (word[i] != '\0') {
        cout << "Character " << i + 1 << ": " << word[i] << endl;
        i++;
    }
    return 0;
}
```

Looping Through a String

```
string name = "Loki";
for (int i = 0; i < name.length(); i++) {
    cout << name[i] << " ";
}
```

Common String Functions

Function	Description	Example
`length()`	Returns size	`s.length()`
`at(i)`	Character at index	`s.at(0)`
`append(str)`	Add to end	`s.append("Man")`
`insert(pos,str)`	Insert substring	`s.insert(2,"abc")`
`erase(pos,len)`	Remove chars	`s.erase(1,3)`
`substr(pos,len)`	Extract substring	`s.substr(2,4)`
`find(str)`	Find position	`s.find("abc")`
`replace(p,l,str)`	Replace part	`s.replace(1,3,"XYZ")`
`compare(str)`	Compare strings	`s.compare("abc")`

Common String Functions

- string s = "Avengers";
- cout << s.length(); // Output: 8

- string s = "Hulk";
- cout << s.at(1); // Output: u

Common String Functions

- string s1 = "Captain ";
- string s2 = "America";
- s1.append(s2);
- cout << s1; // Output: Captain America

- string a = "Black ";
- string b = "Widow";
- string c = a + b;
- cout << c; // Output: Black Widow

Common String Functions

- string s = "ThorOdinson";
- s.insert(4, " ");
- cout << s;

- string s = "SpiderHuman";
- s.erase(6, 5);
- cout << s; // Output: Spider

Common String Functions

- string s = "Doctor Strange";
- string part = s.substr(7, 7);
- cout << part; // Output: Strange

- string s = "Black Panther";
- int index = s.find("Panther");
- cout << index; // Output: 6

Common String Functions

- string s = "Iron Boy";
- s.replace(5, 3, "Man");
- cout << s; // Output: Iron Man

- string a = "Vision";
- string b = "Vision";
- if (a.compare(b) == 0)
- 	cout << "Both are same.";
- else
- 	cout << "Different.";