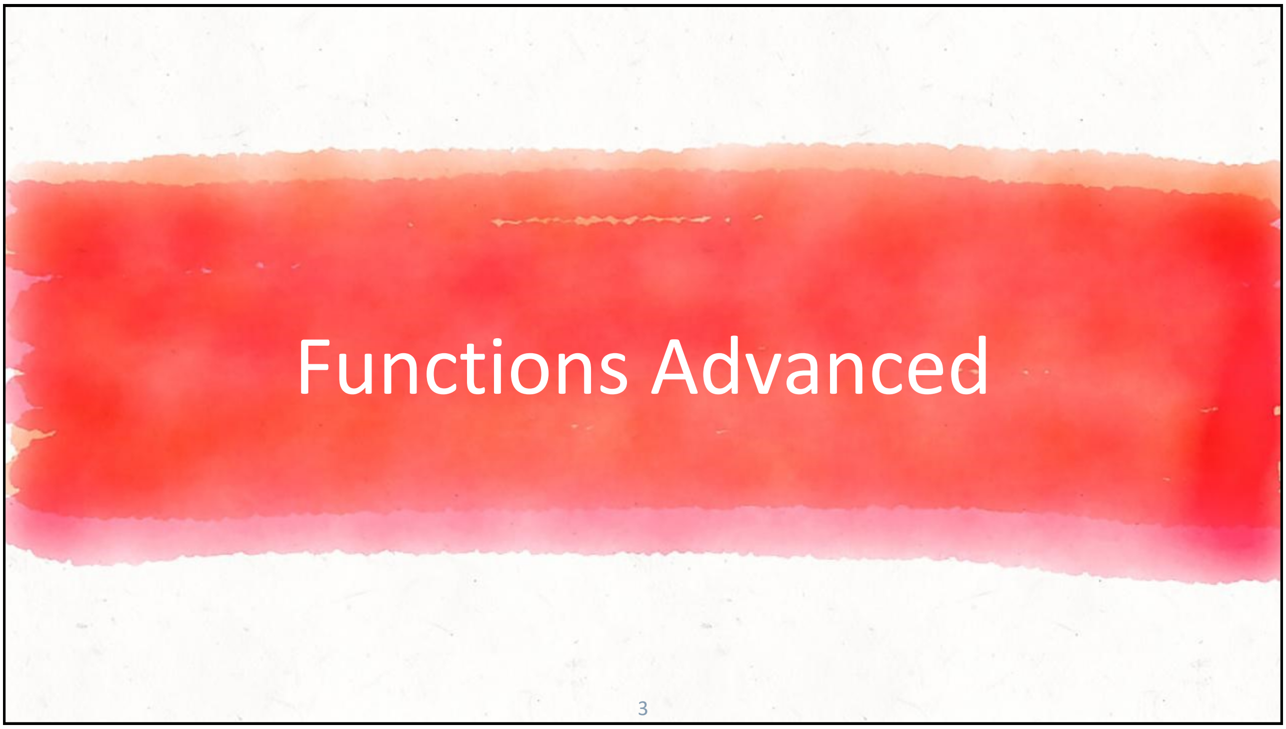
CS4002
Applied Programming

National University of Computer & Emerging Sciences
Islamabad, Pakistan

Ms. Umarah Qaseem

Content

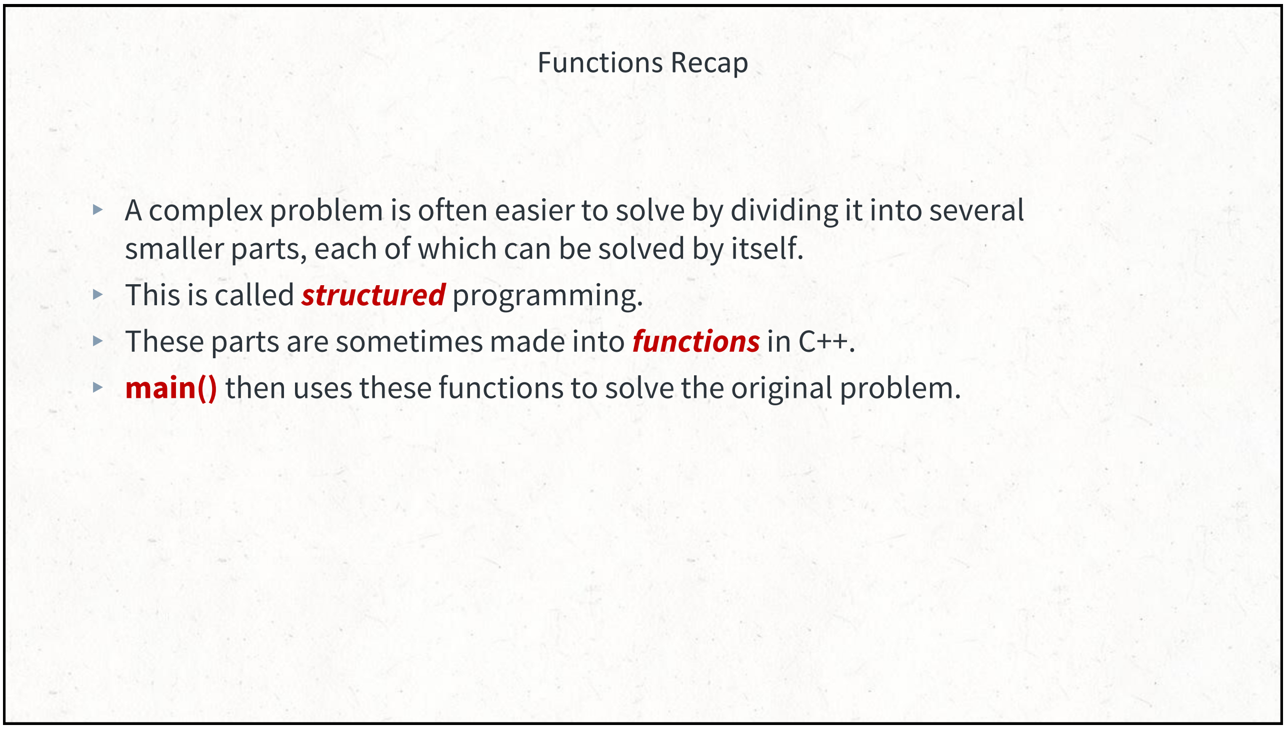
- Functions Advanced
- pass by reference vs value vs pointer
- Pass by pointer as value and reallocating it dynamically
- Return by pointer, const pointer
- Recursion
- Strings
- Structures
- Nested structures
- Structures with pointers
- Structures arrays & structure with functions
- Access modifiers is structure



Functions Advanced

3

3



Functions Recap

- A complex problem is often easier to solve by dividing it into several smaller parts, each of which can be solved by itself.
- This is called ***structured*** programming.
- These parts are sometimes made into ***functions*** in C++.
- **main()** then uses these functions to solve the original problem.

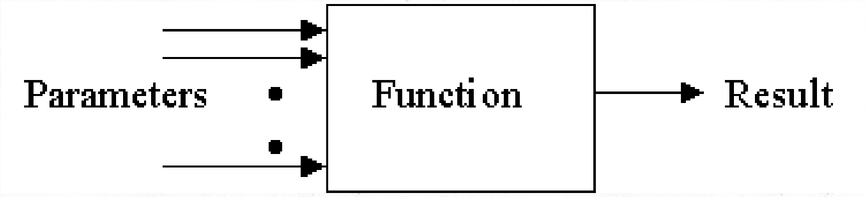
4

Advantages

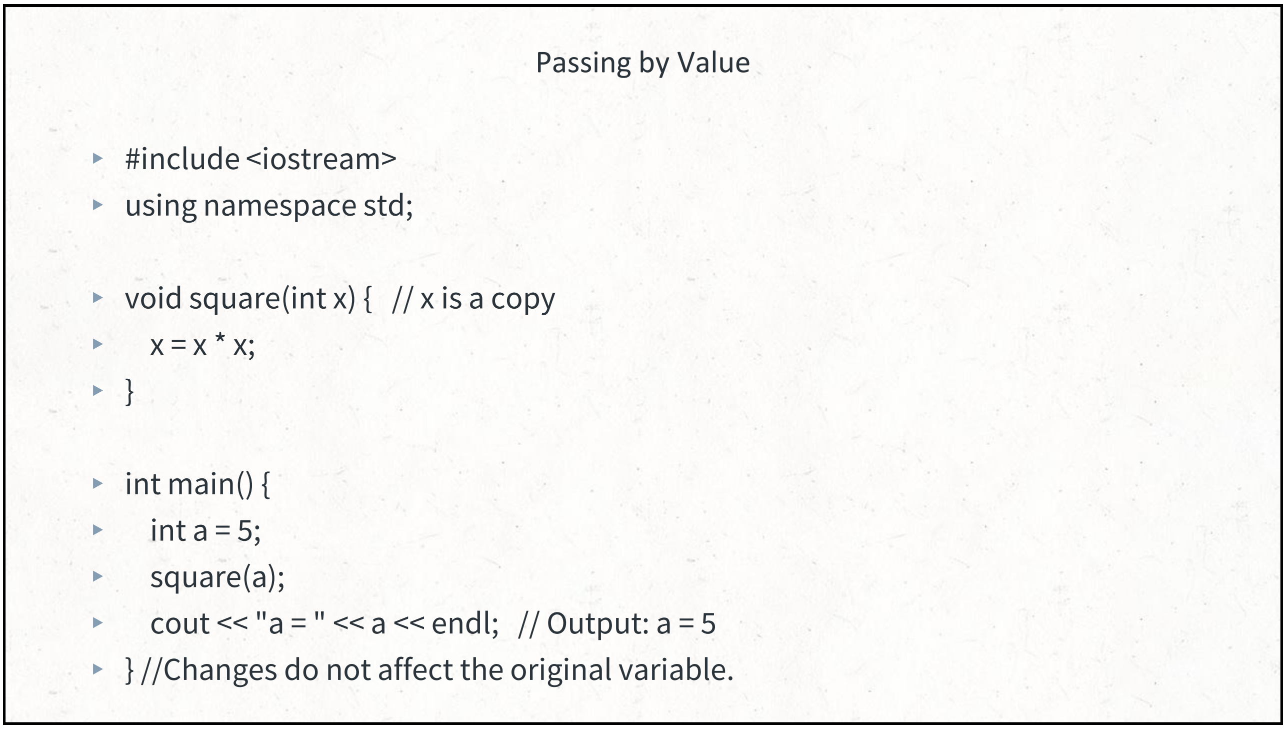
- Functions separate the concept (what is done) from the implementation (how it is done).
- Functions make programs easier to understand.
- Functions can be called several times in the same program, allowing the code to be reused.

5

Function Input & Output



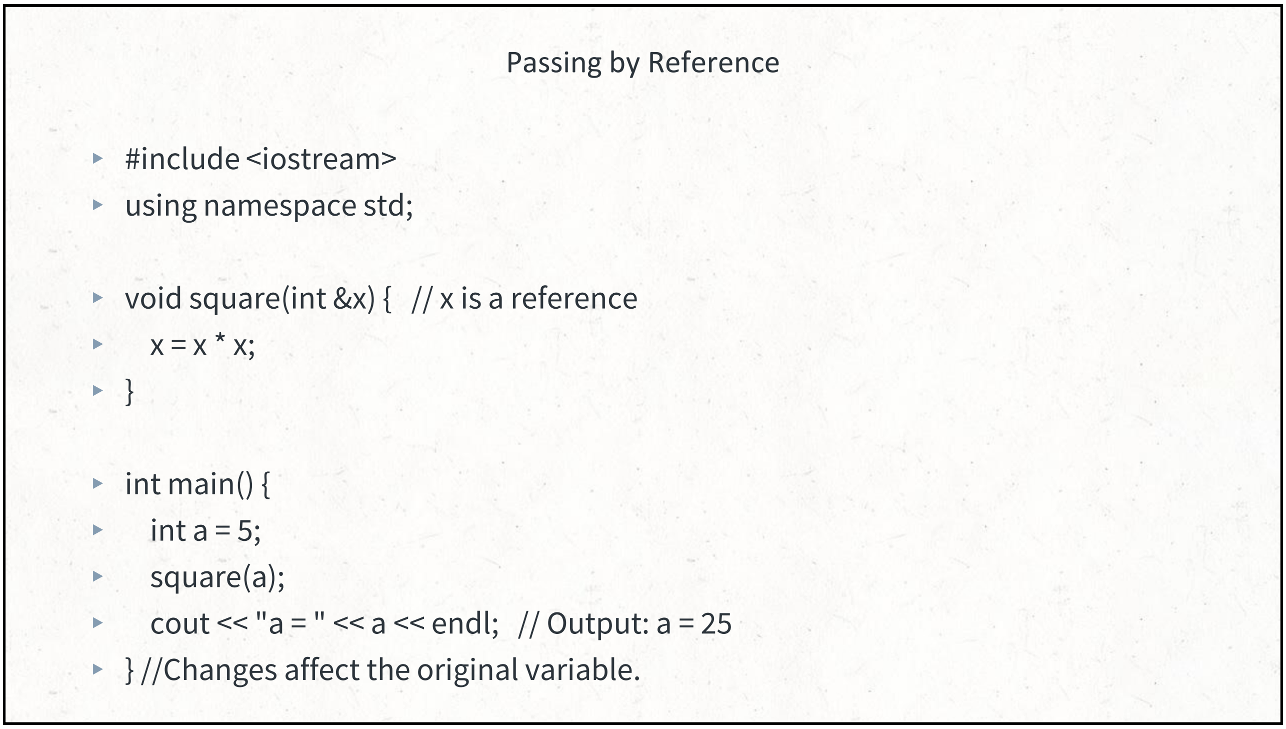
6



Passing by Value

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

void square(int x) {   // x is a copy
    x = x * x;
}

int main() {
    int a = 5;
    square(a);
    cout << "a = " << a << endl;   // Output: a = 5
} //Changes do not affect the original variable.
```

7



Passing by Reference

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

void square(int &x) {   // x is a reference
    x = x * x;
}

int main() {
    int a = 5;
    square(a);
    cout << "a = " << a << endl;   // Output: a = 25
} //Changes affect the original variable.
```

8

Passing by Pointer

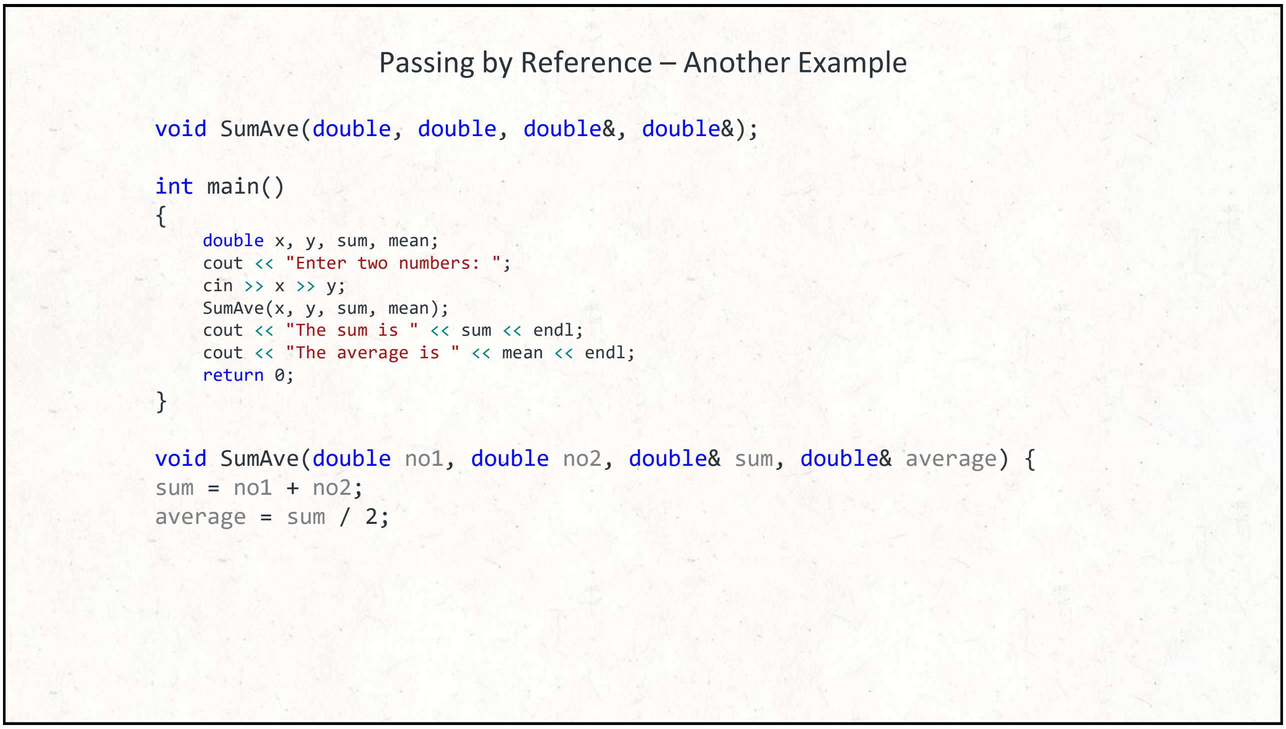
```
#include <iostream>
using namespace std;

void square(int *x) {   // x is a pointer
    *x = (*x) * (*x);
}

int main() {
    int a = 5;
    square(&a);
    cout << "a = " << a << endl;   // Output: a = 25
} //Changes affect the original variable.
```

9

Passing by Reference - Explantation

- The corresponding argument must be a variable.
- The reference of that variable is passed to the function, instead of its value.
- If the function changes the parameter value, the change will be reflected in the corresponding argument, since they share the same memory location.
- To have a function with multiple outputs, we have to use pass by reference.
- We use **&** to denote a parameter that is passed by reference: `<type>&`

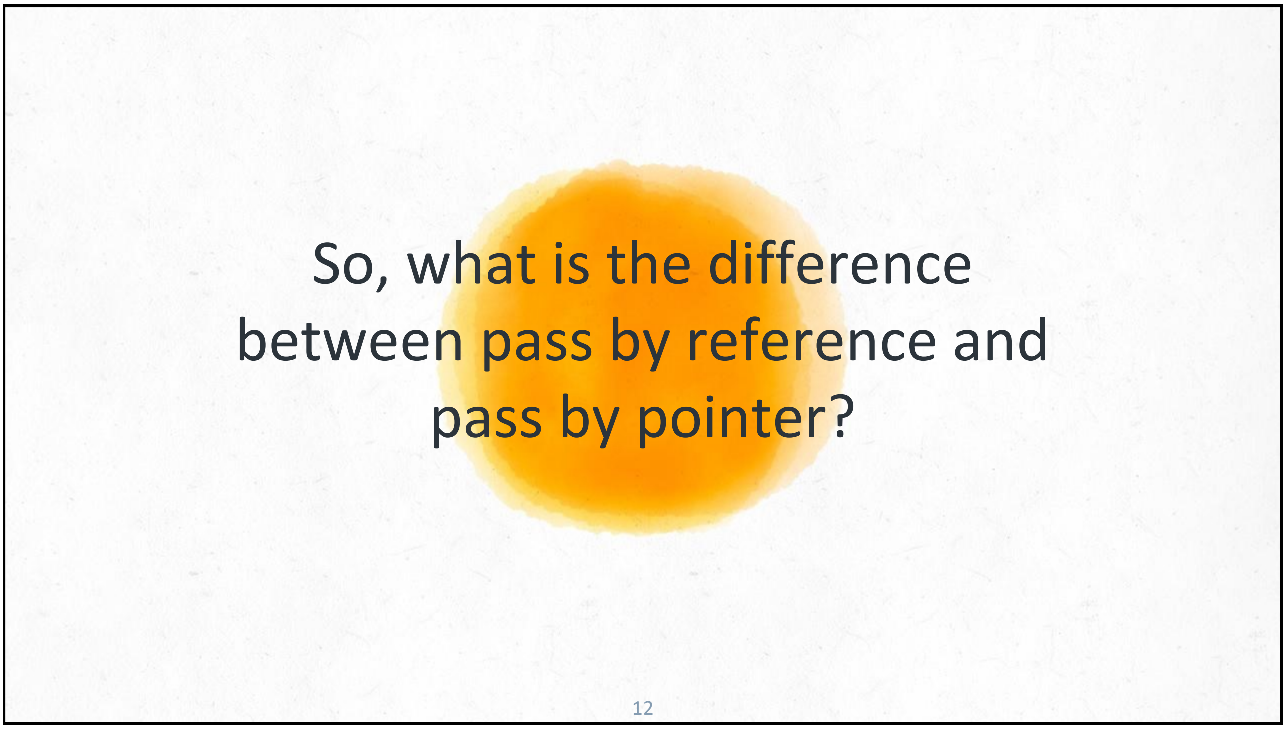
10



Passing by Reference – Another Example

```
void SumAve(double, double, double&, double&);

int main()
{
    double x, y, sum, mean;
    cout << "Enter two numbers: ";
    cin >> x >> y;
    SumAve(x, y, sum, mean);
    cout << "The sum is " << sum << endl;
    cout << "The average is " << mean << endl;
    return 0;
}

void SumAve(double no1, double no2, double& sum, double& average) {
sum = no1 + no2;
average = sum / 2;
```

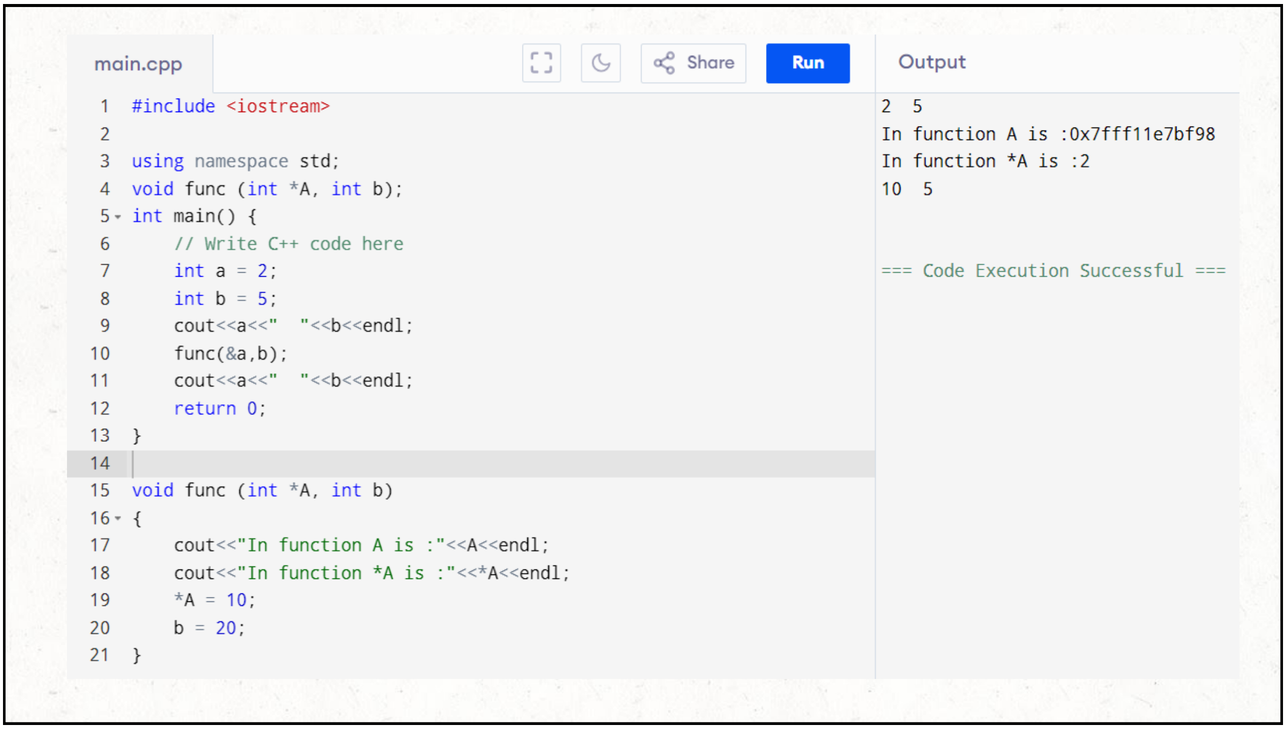
11



So, what is the difference between pass by reference and pass by pointer?

12

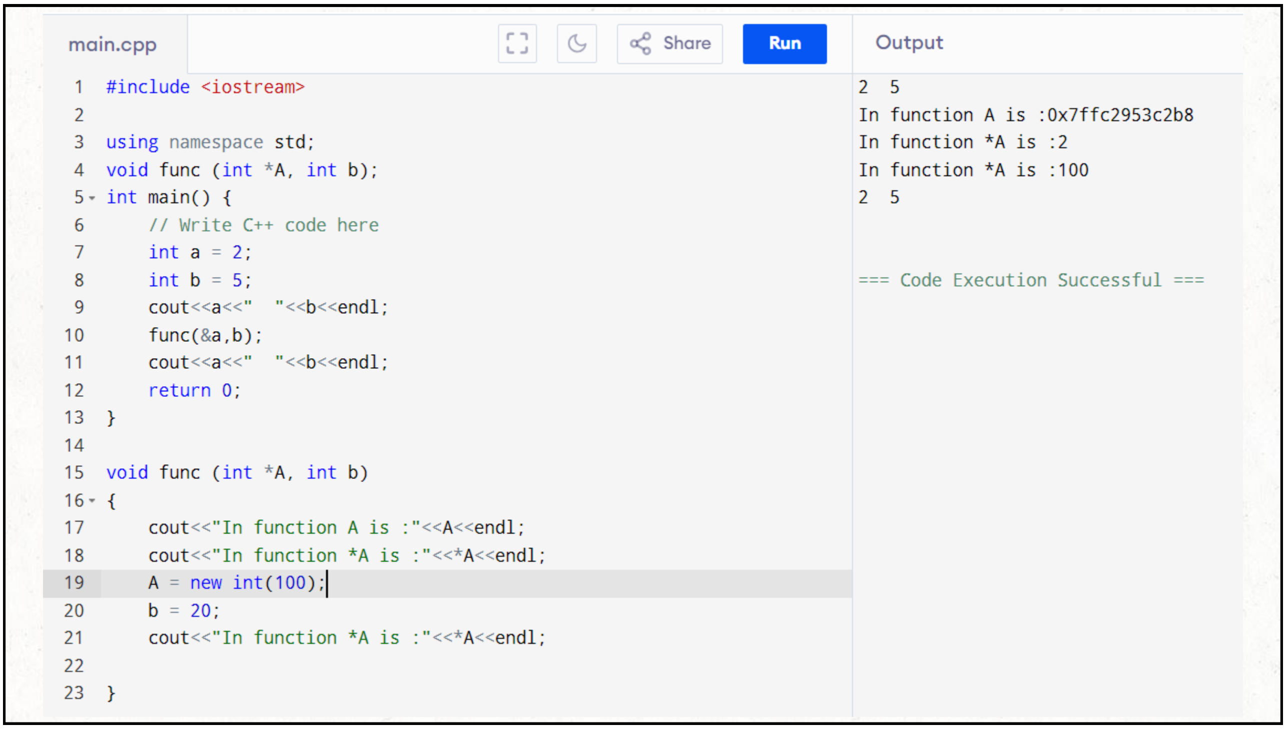
12



```cpp
#include <iostream>

using namespace std;
void func (int *A, int b);
int main() {
    // Write C++ code here
    int a = 2;
    int b = 5;
    cout<<a<<"  "<<b<<endl;
    func(&a,b);
    cout<<a<<"  "<<b<<endl;
    return 0;
}

void func (int *A, int b)
{
    cout<<"In function A is :"<<A<<endl;
    cout<<"In function *A is :"<<*A<<endl;
    *A = 10;
    b = 20;
}
```

```
2  5
In function A is :0x7fff11e7bf98
In function *A is :2
10  5


=== Code Execution Successful ===
```

13



```cpp
#include <iostream>

using namespace std;
void func (int *A, int b);
int main() {
    // Write C++ code here
    int a = 2;
    int b = 5;
    cout<<a<<"  "<<b<<endl;
    func(&a,b);
    cout<<a<<"  "<<b<<endl;
    return 0;
}

void func (int *A, int b)
{
    cout<<"In function A is :"<<A<<endl;
    cout<<"In function *A is :"<<*A<<endl;
    A = new int(100);
    b = 20;
    cout<<"In function *A is :"<<*A<<endl;

}
```

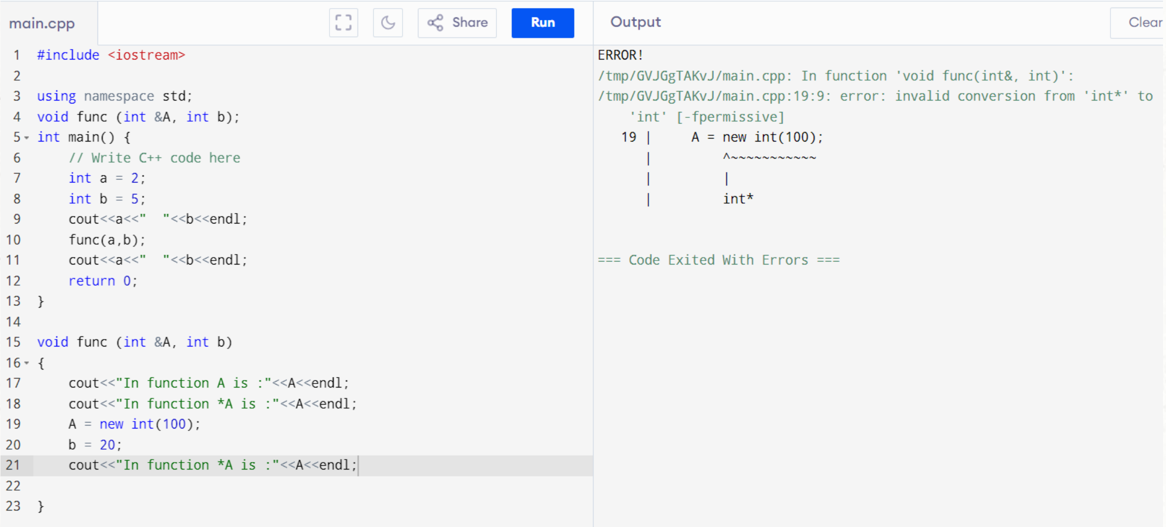
```
2  5
In function A is :0x7ffc2953c2b8
In function *A is :2
In function *A is :100
2  5


=== Code Execution Successful ===
```

14

What did you learn?

- The pointer was passed by value!
- That means that a copy of the address was made and sent to a new variable!
- So when you make the change it was reflected in main program because you were making change at the address.
- BUT!
- If you change the address, then the change won’t be reflected any more in the main program!
- Why? because the pointer itself was passed by value (a copy).
- In easy words, A duplicate pointer with same address was made.

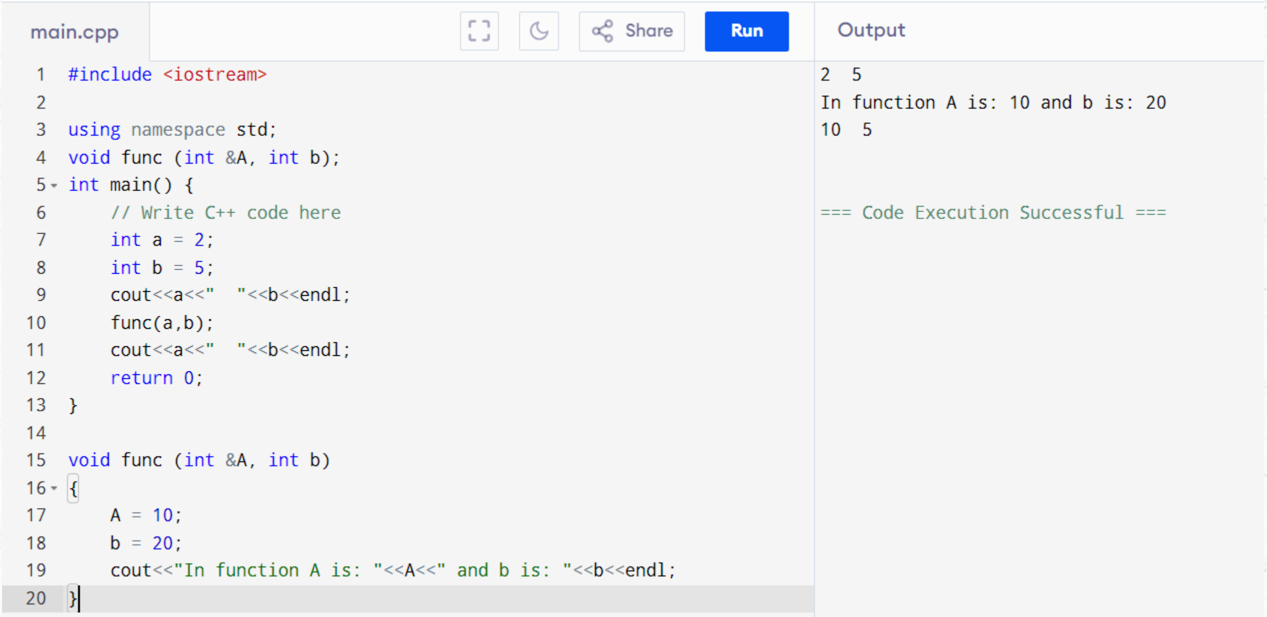
Do the same thing with pass by reference and you get an error!



```cpp
1  #include <iostream>
2
3  using namespace std;
4  void func (int &A, int b);
5  int main() {
6      // Write C++ code here
7      int a = 2;
8      int b = 5;
9      cout<<a<<"  "<<b<<endl;
10     func(a,b);
11     cout<<a<<"  "<<b<<endl;
12     return 0;
13 }
14
15 void func (int &A, int b)
16 {
17     cout<<"In function A is :"<<A<<endl;
18     cout<<"In function *A is :"<<A<<endl;
19     A = new int(100);
20     b = 20;
21     cout<<"In function *A is :"<<A<<endl;
22
23 }
```

```
ERROR!
/tmp/GVJGgTAKvJ/main.cpp: In function 'void func(int&, int)':
/tmp/GVJGgTAKvJ/main.cpp:19:9: error: invalid conversion from 'int*' to
    'int' [-fpermissive]
   19 |     A = new int(100);
      |         ^~~~~~~~~~~~
      |         |
      |         int*


=== Code Exited With Errors ===
```

Correct Code



```cpp
#include <iostream>

using namespace std;
void func (int &A, int b);
int main() {
    // Write C++ code here
    int a = 2;
    int b = 5;
    cout<<a<<"  "<<b<<endl;
    func(a,b);
    cout<<a<<"  "<<b<<endl;
    return 0;
}

void func (int &A, int b)
{
    A = 10;
    b = 20;
    cout<<"In function A is: "<<A<<" and b is: "<<b<<endl;
}
```

```
2  5
In function A is: 10 and b is: 20
10  5


=== Code Execution Successful ===
```

17

Arrays as Function Parameters

18

Arrays as Function Parameters

- **`void init(float A[], int arraySize);`**
  `void init(float *A, int arraySize);`

- Are identical function prototypes!
- Pointer is passed by value
- I.e. caller copies the *value* of a pointer to **`float`** into the parameter **`A`**
- Called function can reference *through* that pointer to reach thing pointed to

19

Result

- Even though all arguments are passed *by value* to functions …
- … pointers allow functions to assign back to data of caller

- Arrays are pointers passed by value

20

```
//^^^^^^^^^^^^^^^^^^^^^1^^^^^^^^^^^^^^^^^^^^^^^^
void printArrayElements(int[], int);
void printArrayElementsWithPtr(int*, int);
//---------------------------------------
void main()
{
	const int size = 10;
	int myArray[size] = { 32,43,23,65,54,4,-1,76,67,8, };
	cout << "Print with Array argument: " << endl;
	printArrayElements(myArray, size);
	cout << endl<<"Print with pointer argument: " << endl;
	printArrayElementsWithPtr(myArray, size);
	cout << endl;
}
//^^^^^^^^^^^^^^^^^^^^^2^^^^^^^^^^^^^^^^^^^^^^^^
//------------------------------------
void printArrayElements(int Array[], int arraySize)
{
	for (int index = 0; index < arraySize; index++)
	cout << "  " << Array[index];
}
void printArrayElementsWithPtr(int* Array, int arraySize)
{
	for (int index = 0; index < arraySize; index++)
	cout << "  " << Array[index];
}
```

21

Safety Note – const

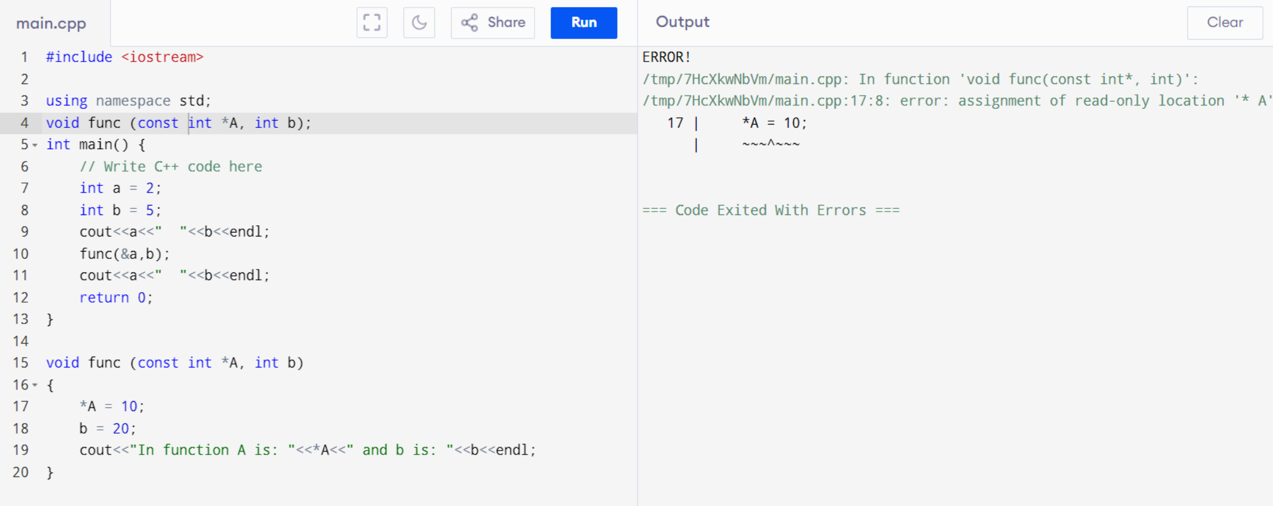
- When passing arrays to functions, *it is recommended to* specify **`const`** if you don't want function changing the value of any elements
- Reason:– you don't know whether your function would pass array to another before returning to you
  - Exception – many software packages don't specify **`const`** in their own headers, so you can't either!

22

Arrays used as constant input

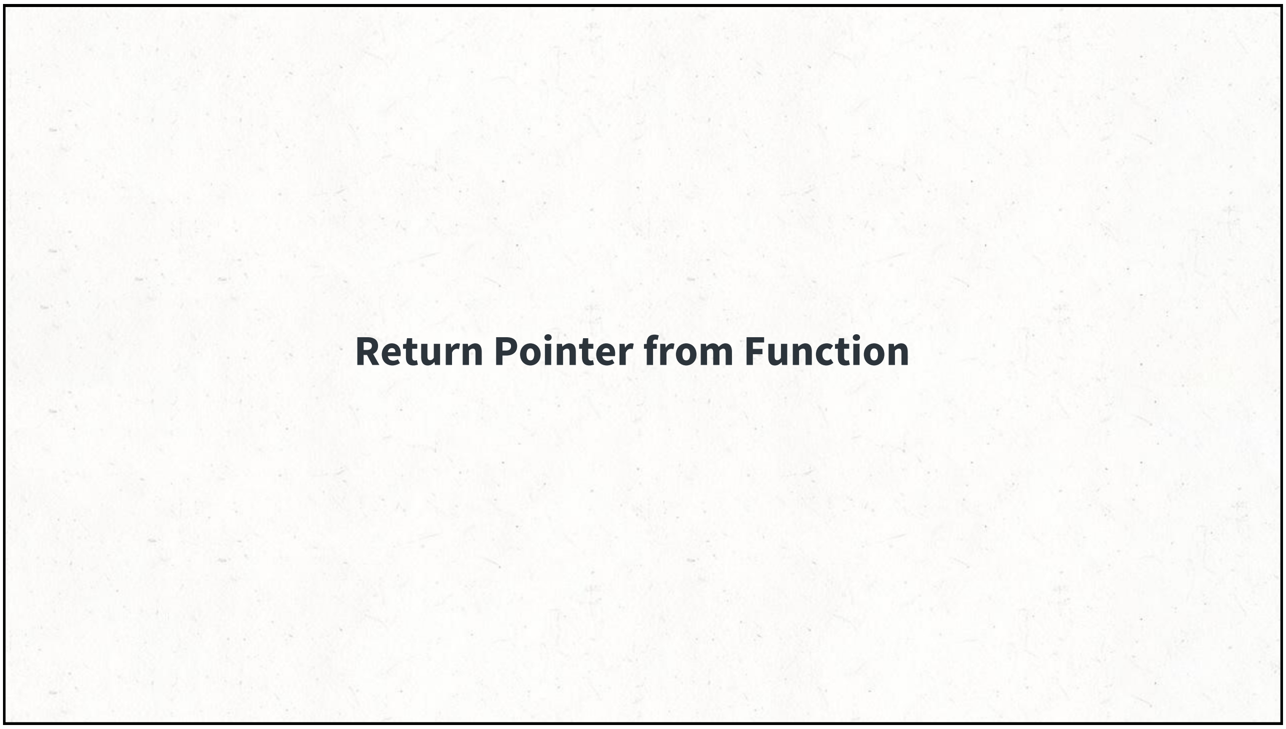
- What happens when want to use array only as input? We can't pass it by value…
  - ○ `void large (int size, const int arry1[], const int arry2[], int arry3[]);`
- We can protect array arguments by putting const in front of them in prototype and function definition

Const Example



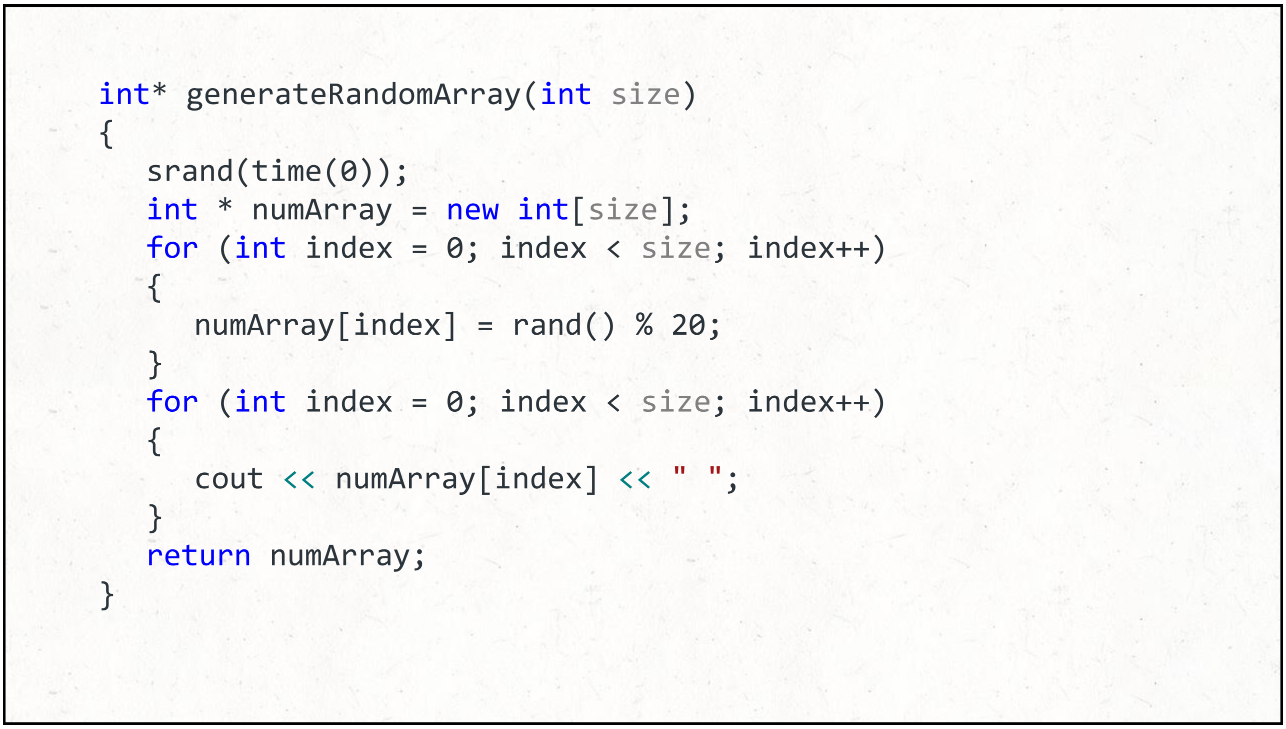
```
1  #include <iostream>
2
3  using namespace std;
4  void func (const int *A, int b);
5  int main() {
6      // Write C++ code here
7      int a = 2;
8      int b = 5;
9      cout<<a<<"  "<<b<<endl;
10     func(&a,b);
11     cout<<a<<"  "<<b<<endl;
12     return 0;
13 }
14
15 void func (const int *A, int b)
16 {
17     *A = 10;
18     b = 20;
19     cout<<"In function A is: "<<*A<<" and b is: "<<b<<endl;
20 }
```

```
ERROR!
/tmp/7HcXkwNbVm/main.cpp: In function 'void func(const int*, int)':
/tmp/7HcXkwNbVm/main.cpp:17:8: error: assignment of read-only location '* A'
   17 |     *A = 10;
      |     ~~~^~~~


=== Code Exited With Errors ===
```



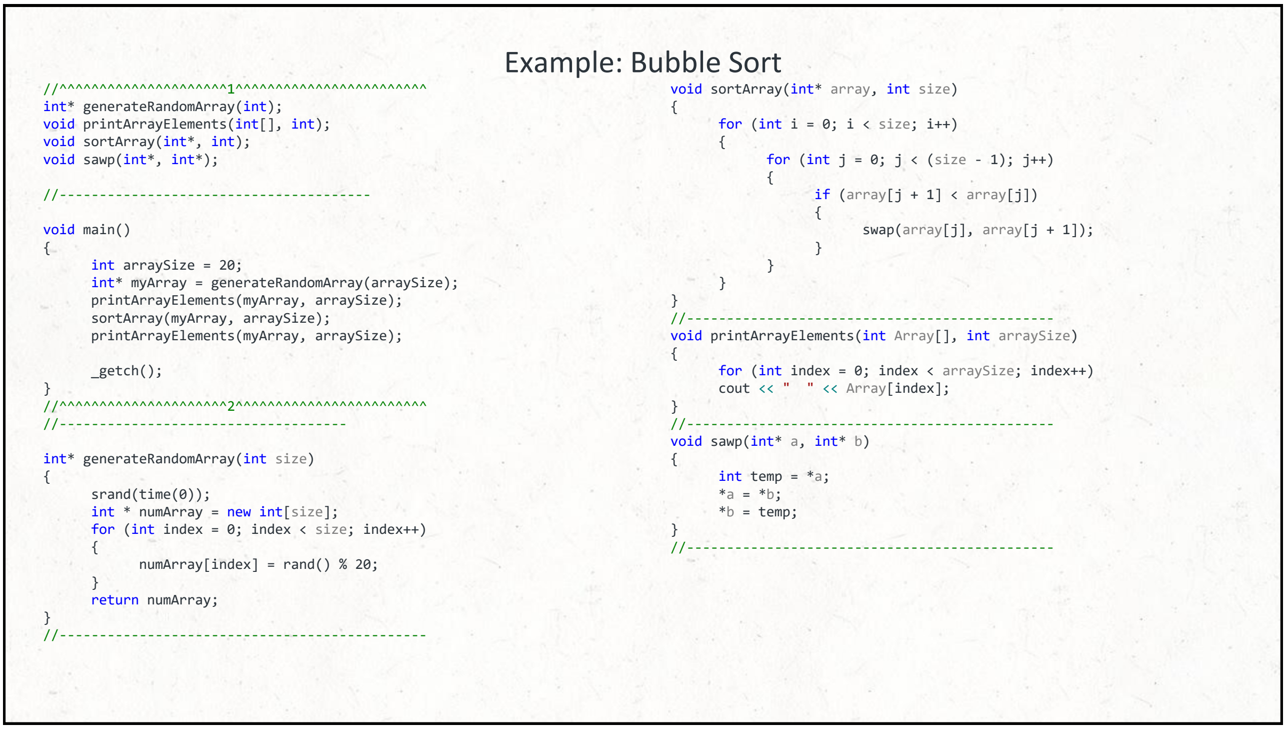
Return Pointer from Function

25



```
int* generateRandomArray(int size)
{
   srand(time(0));
   int * numArray = new int[size];
   for (int index = 0; index < size; index++)
   {
      numArray[index] = rand() % 20;
   }
   for (int index = 0; index < size; index++)
   {
      cout << numArray[index] << " ";
   }
   return numArray;
}
```

26

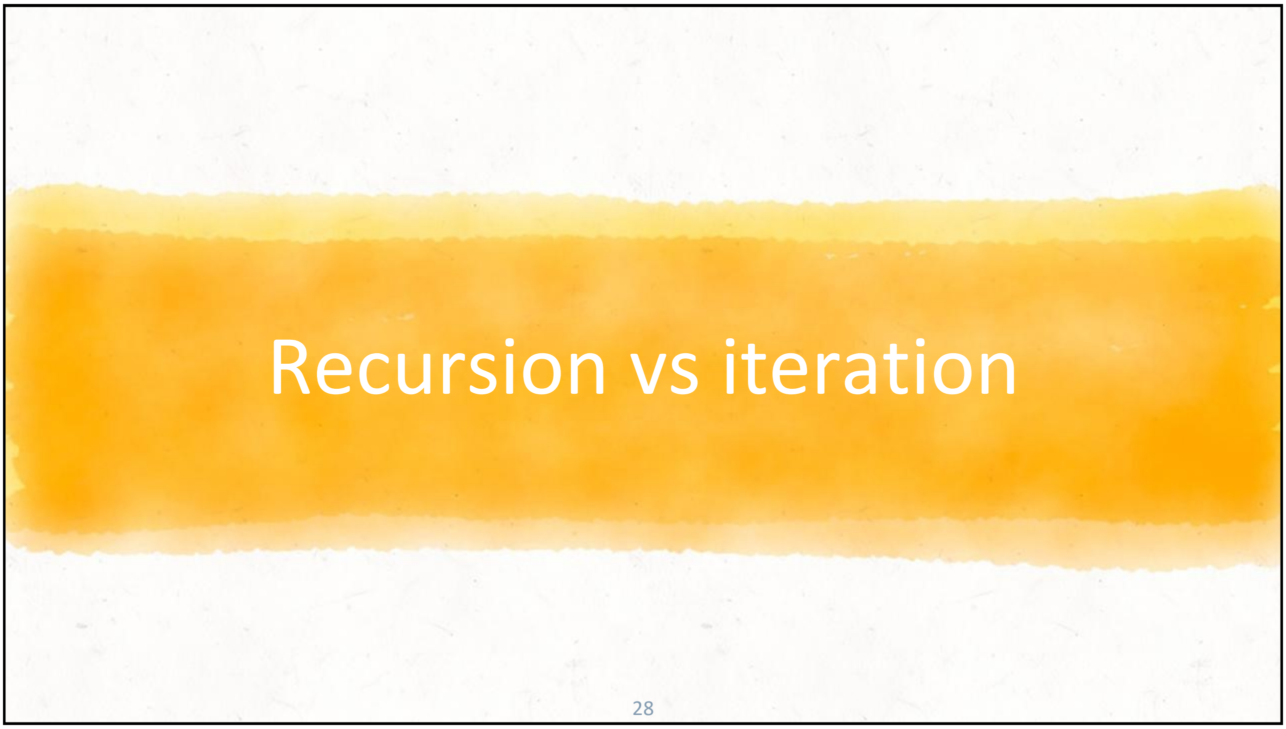


Example: Bubble Sort

```
//^^^^^^^^^^^^^^^^^^^^^1^^^^^^^^^^^^^^^^^^^^^^^^
int* generateRandomArray(int);
void printArrayElements(int[], int);
void sortArray(int*, int);
void sawp(int*, int*);

//---------------------------------------

void main()
{
      int arraySize = 20;
      int* myArray = generateRandomArray(arraySize);
      printArrayElements(myArray, arraySize);
      sortArray(myArray, arraySize);
      printArrayElements(myArray, arraySize);

      _getch();
}
//^^^^^^^^^^^^^^^^^^^^^2^^^^^^^^^^^^^^^^^^^^^^^^
//------------------------------------

int* generateRandomArray(int size)
{
      srand(time(0));
      int * numArray = new int[size];
      for (int index = 0; index < size; index++)
      {
            numArray[index] = rand() % 20;
      }
      return numArray;
}
//----------------------------------------------
```

```
void sortArray(int* array, int size)
{
      for (int i = 0; i < size; i++)
      {
            for (int j = 0; j < (size - 1); j++)
            {
                  if (array[j + 1] < array[j])
                  {
                        swap(array[j], array[j + 1]);
                  }
            }
      }
}
//----------------------------------------------
void printArrayElements(int Array[], int arraySize)
{
      for (int index = 0; index < arraySize; index++)
      cout << "  " << Array[index];
}
//----------------------------------------------
void sawp(int* a, int* b)
{
      int temp = *a;
      *a = *b;
      *b = temp;
}
//----------------------------------------------
```

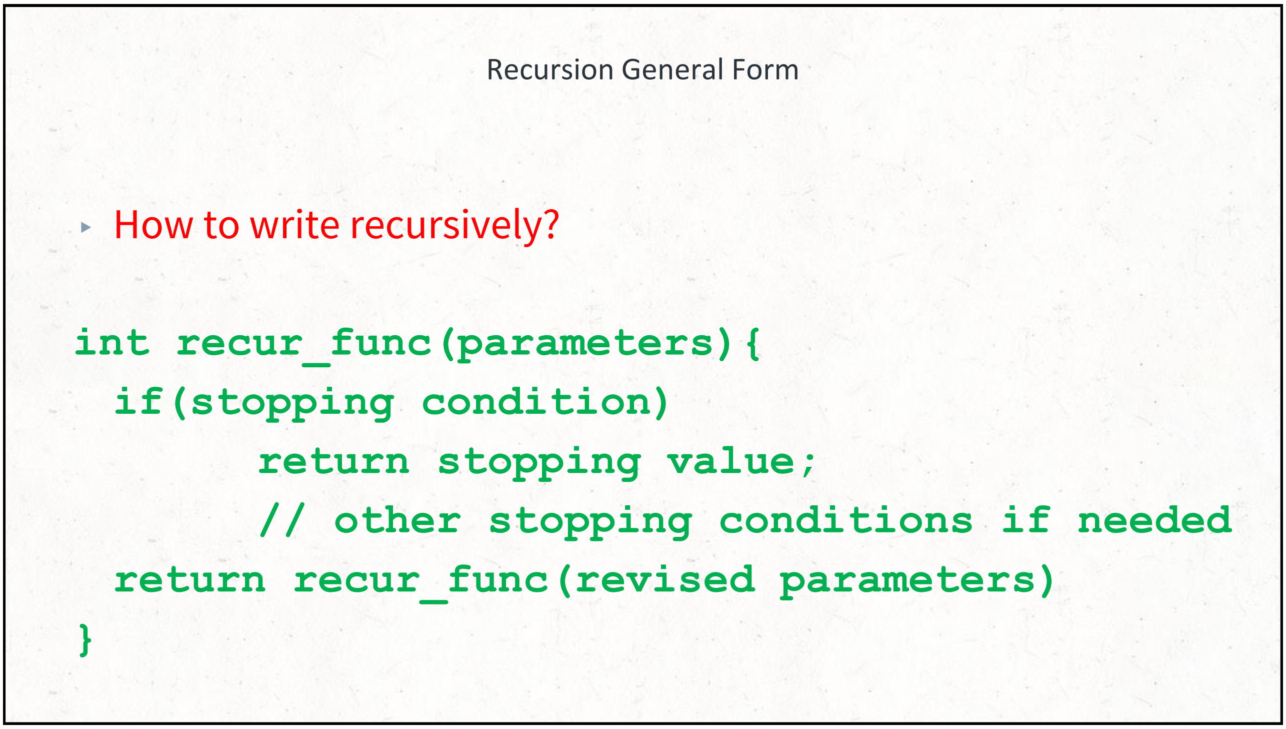
27



Recursion vs iteration

28

28

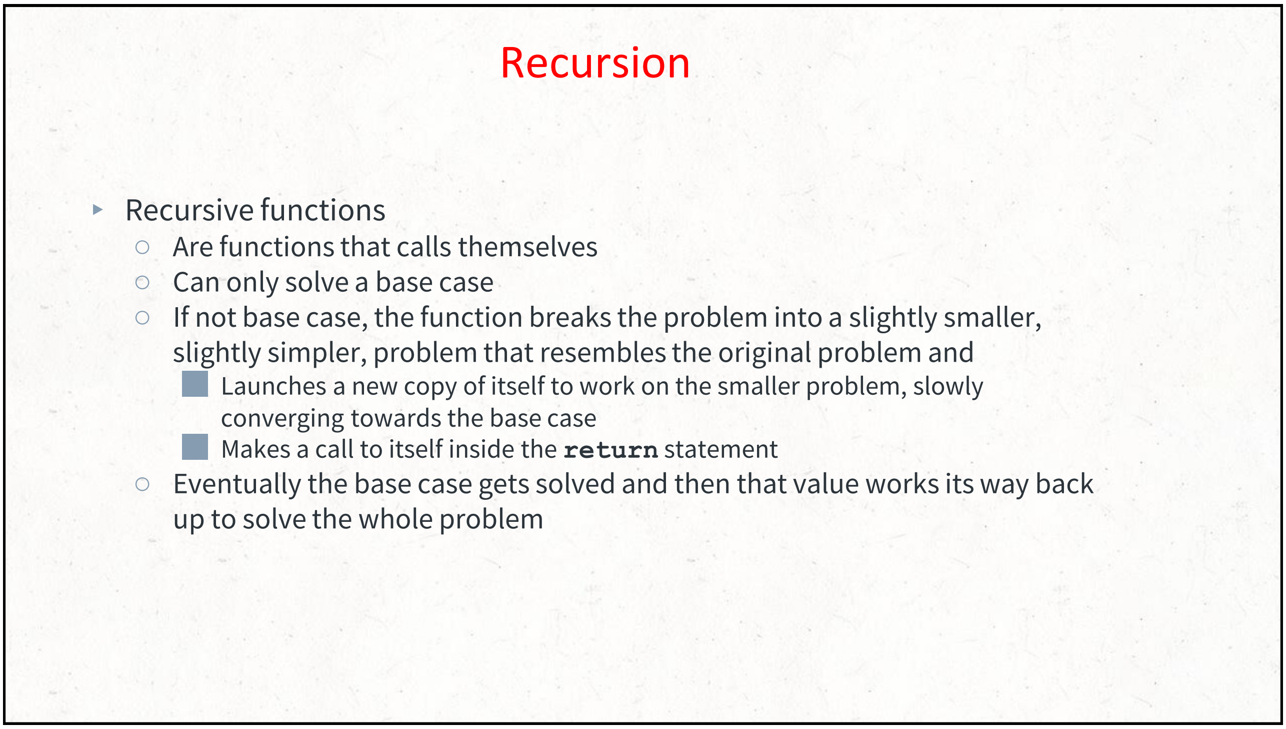


Recursion General Form

- How to write recursively?

```
int recur_func(parameters){
  if(stopping condition)
       return stopping value;
       // other stopping conditions if needed
  return recur_func(revised parameters)
}
```

29



Recursion

- Recursive functions
  - Are functions that calls themselves
  - Can only solve a base case
  - If not base case, the function breaks the problem into a slightly smaller, slightly simpler, problem that resembles the original problem and
    - Launches a new copy of itself to work on the smaller problem, slowly converging towards the base case
    - Makes a call to itself inside the `return` statement
  - Eventually the base case gets solved and then that value works its way back up to solve the whole problem

30

Recursion

- Example: factorial

  *n! = n * (n – 1) * (n – 2) * … * 1*

  - Recursive relationship (*n! = n * (n – 1)!*)

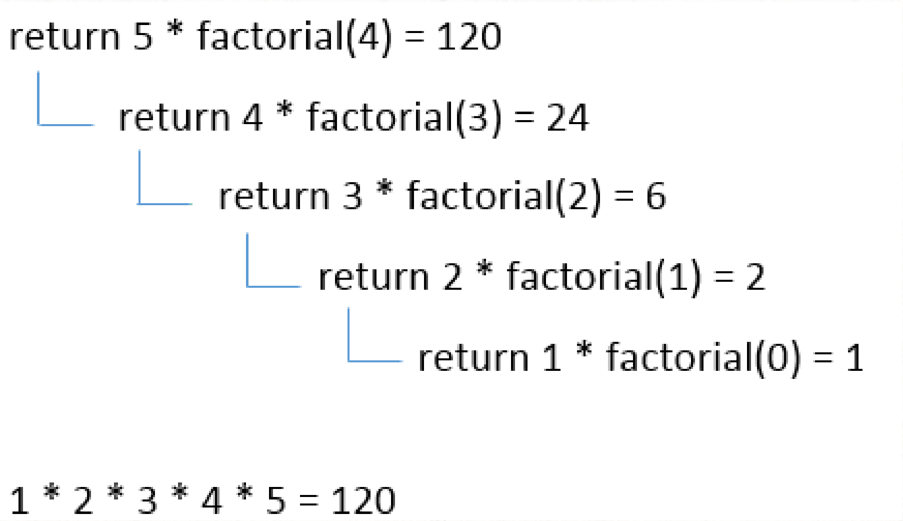
    *5! = 5 * 4!*

    *4! = 4 * 3!…*

  - Base case (*1! = 0! = 1*)

```
//^^^^^^^^^^^^^^^^^^^^^1^^^^^^^^^^^^^^^^^^^^^^^^
int factorial(int);
//---------------------------------------

void main()
{
    int myNumber = 5;
    cout << "Factorial of Number " << myNumber << " is : " << factorial(myNumber);

    _getch();
}
//^^^^^^^^^^^^^^^^^^^^^2^^^^^^^^^^^^^^^^^^^^^^^^
//------------------------------------
int factorial(int number)
{
   if (number < 1)
      return 1;
   else
      return number * factorial(number - 1);
}
```

Dry Run for Factorial(5)



return 5 * factorial(4) = 120

return 4 * factorial(3) = 24

return 3 * factorial(2) = 6

return 2 * factorial(1) = 2

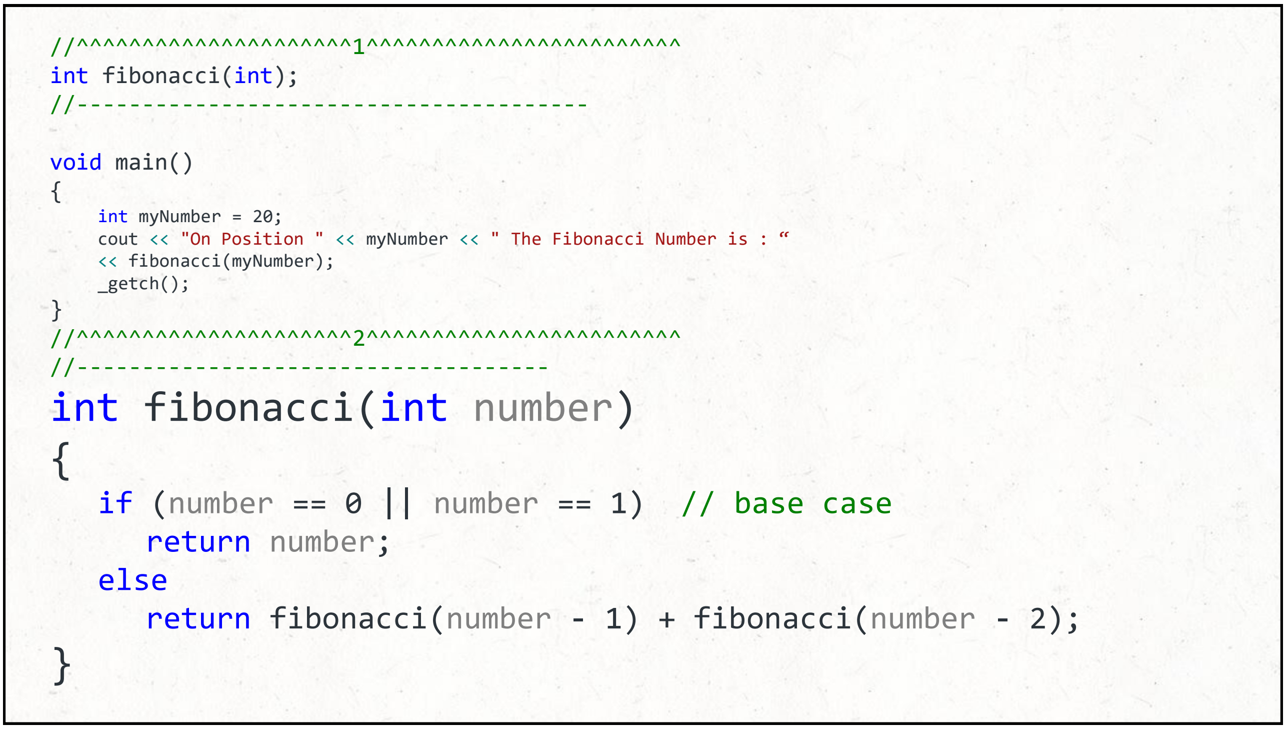
return 1 * factorial(0) = 1

1 * 2 * 3 * 4 * 5 = 120

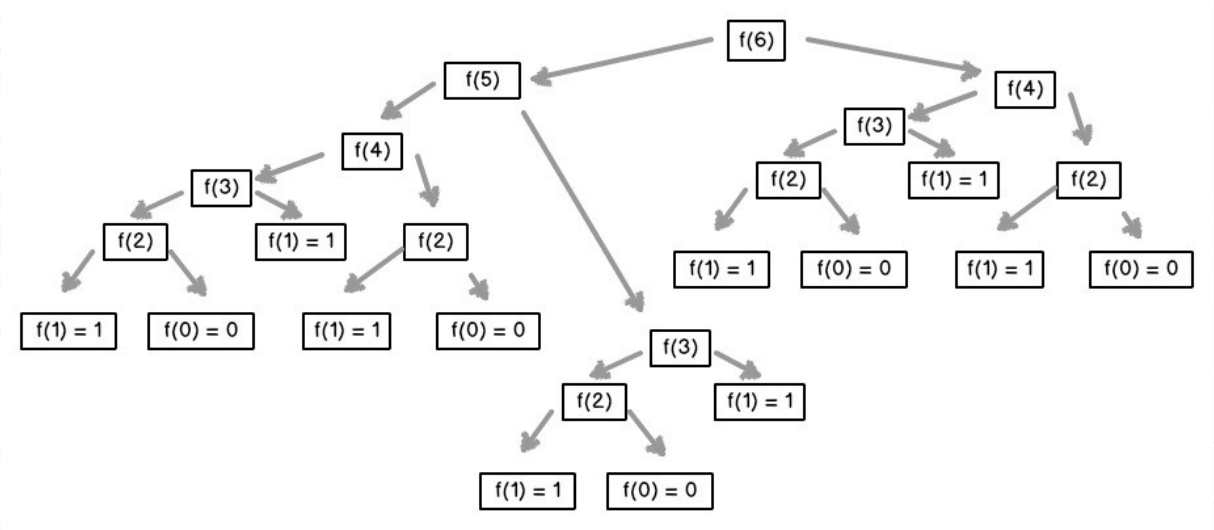
The Fibonacci Series

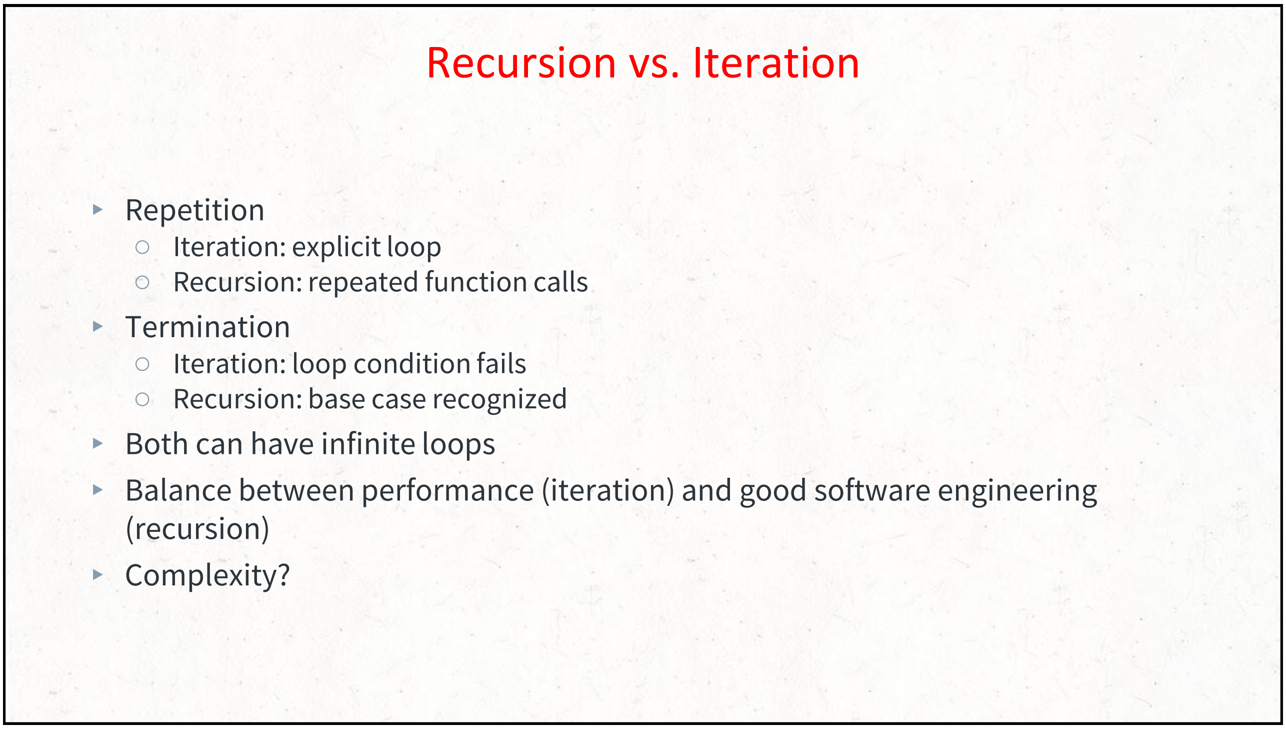
- Fibonacci series: 0, 1, 1, 2, 3, 5, 8…
  - Each number sum of two previous ones
  - Example of a recursive formula:

```
fib(n) = fib(n-1) + fib(n-2)
```



```
//^^^^^^^^^^^^^^^^^^^^^^1^^^^^^^^^^^^^^^^^^^^^^^^^
int fibonacci(int);
//----------------------------------------

void main()
{
    int myNumber = 20;
    cout << "On Position " << myNumber << " The Fibonacci Number is : “
    << fibonacci(myNumber);
    _getch();
}
//^^^^^^^^^^^^^^^^^^^^^^2^^^^^^^^^^^^^^^^^^^^^^^^^
//-------------------------------------
int fibonacci(int number)
{
    if (number == 0 || number == 1)  // base case
        return number;
    else
        return fibonacci(number - 1) + fibonacci(number - 2);
}
```

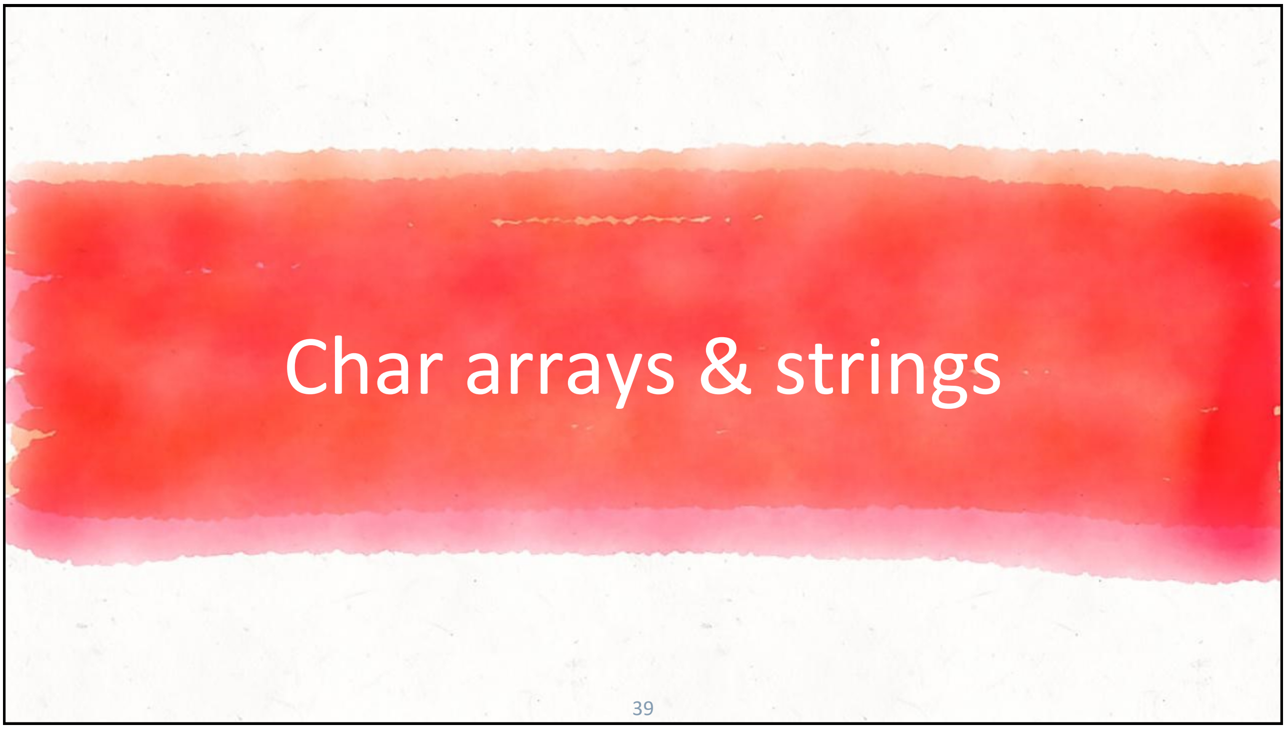




Recursion vs. Iteration

- Repetition
  - Iteration: explicit loop
  - Recursion: repeated function calls
- Termination
  - Iteration: loop condition fails
  - Recursion: base case recognized
- Both can have infinite loops
- Balance between performance (iteration) and good software engineering (recursion)
- Complexity?

37



Char arrays & strings

39

39

Char arrays

- Array of characters

    char test[6] = {‘h’, ’e’, ‘l’, ‘l’, ‘o’, ‘\0’};

    char test[6] = “hello”;

Null character

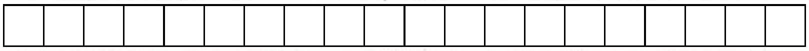
Char arrays

- char arr1[5] = {'h', 'e', 'l', 'l', 'o'};
- char arr2[6] = "hello";   // notice the extra space for '\0’

- arr1 does not have a null character '\0' automatically (unless you put it)
- arr2 does have a null character at the end because you wrote "hello" (string literal). It is stored as: 'h' 'e' 'l' 'l' 'o' '\0’

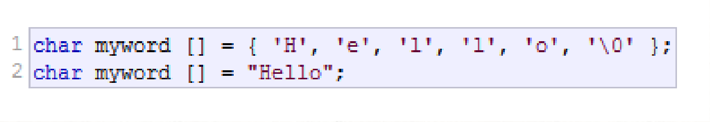
- If you define char arr[5] = "hello"; → ❌ Error, because "hello" needs 6 characters (5 letters + '\0').

Char arrays



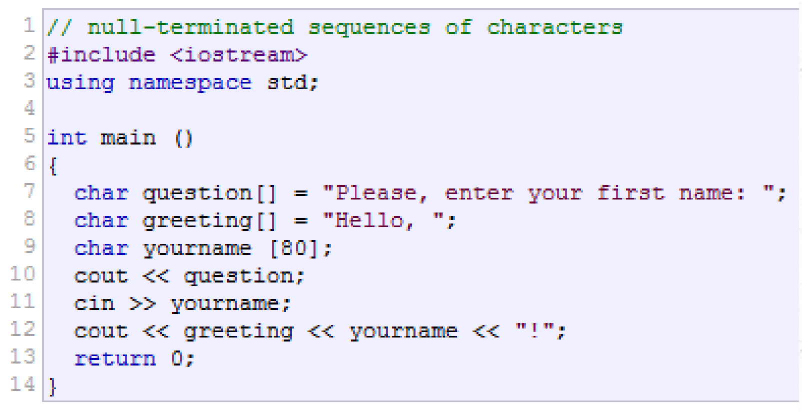
char x[20;]

H	e	l	l	o	\0	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨	▨
M	e	r	r	y		C	h	r	i	s	t	m	a	s	\0	▨	▨	▨	▨

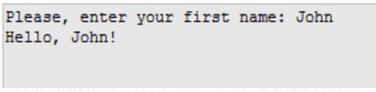


```
char myword [] = { 'H', 'e', 'l', 'l', 'o', '\0' };
char myword [] = "Hello";
```

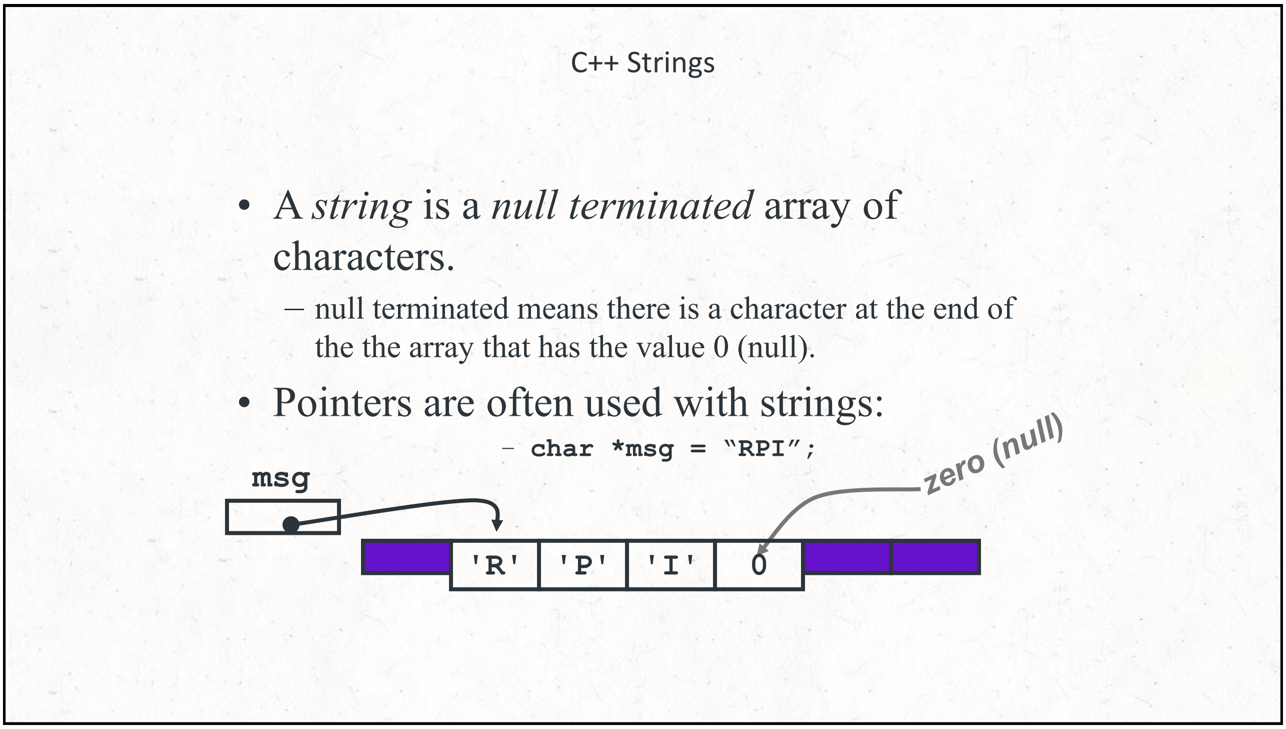
Char arrays



```
// null-terminated sequences of characters
#include <iostream>
using namespace std;

int main ()
{
  char question[] = "Please, enter your first name: ";
  char greeting[] = "Hello, ";
  char yourname [80];
  cout << question;
  cin >> yourname;
  cout << greeting << yourname << "!";
  return 0;
}
```



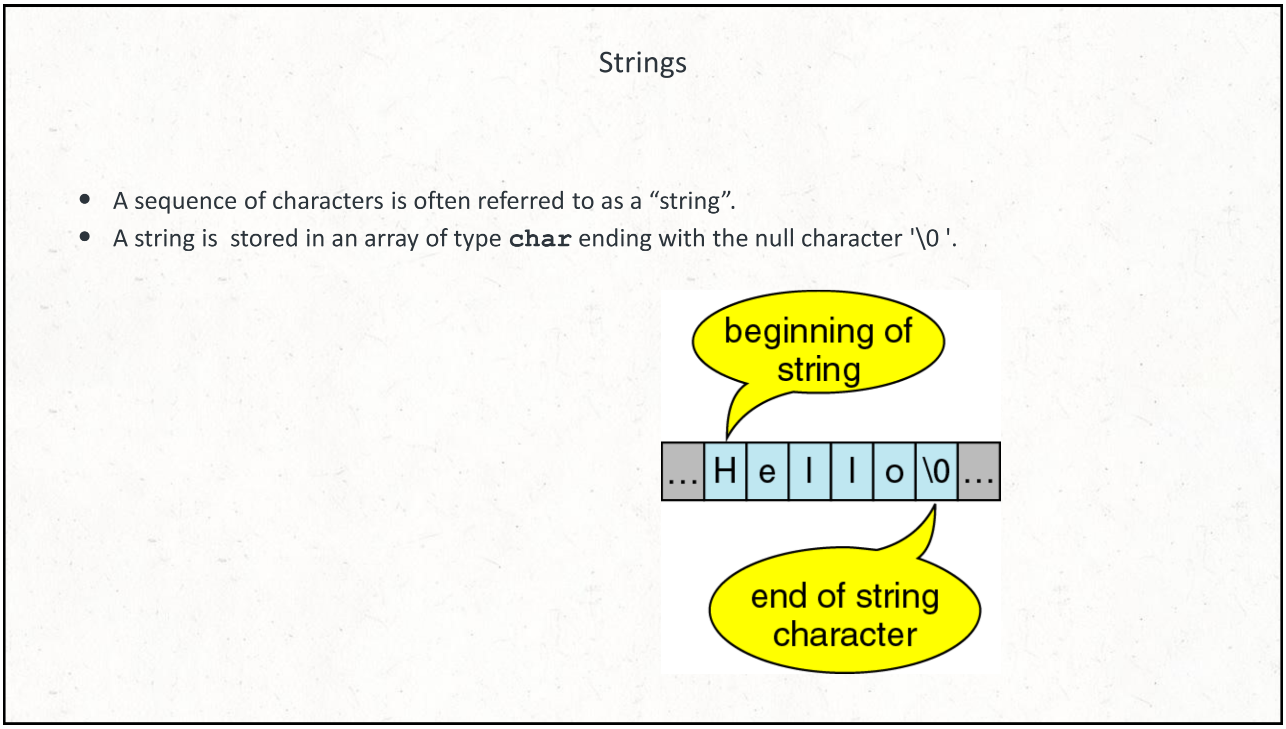
```
Please, enter your first name: John
Hello, John!
```



C++ Strings

- A *string* is a *null terminated* array of characters.
  - null terminated means there is a character at the end of the the array that has the value 0 (null).
- Pointers are often used with strings:
  - `char *msg = "RPI";`

44



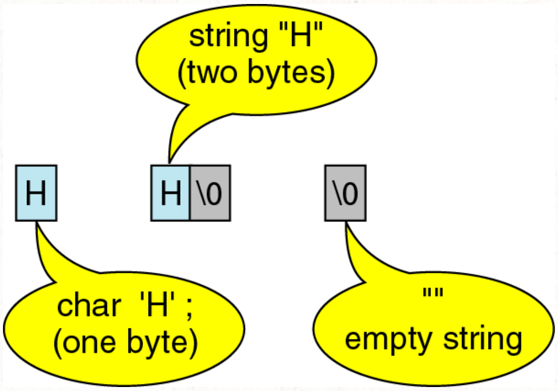
Strings

- A sequence of characters is often referred to as a “string”.
- A string is stored in an array of type **`char`** ending with the null character '\0 '.

45

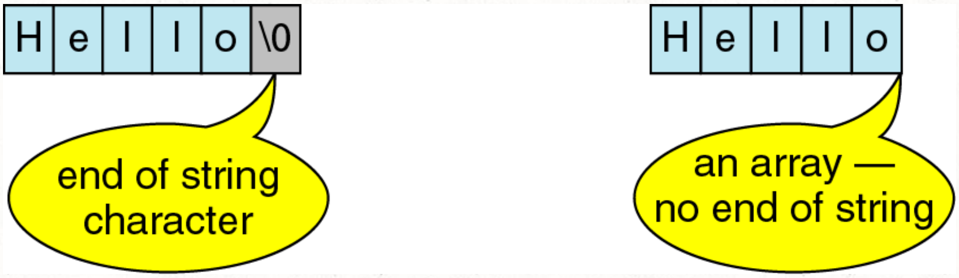
Strings

A string containing a single character takes up 2 bytes of storage.



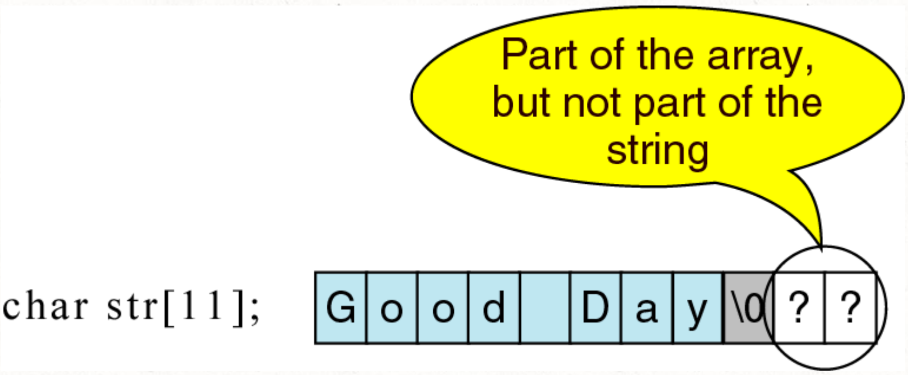
46

Strings



47

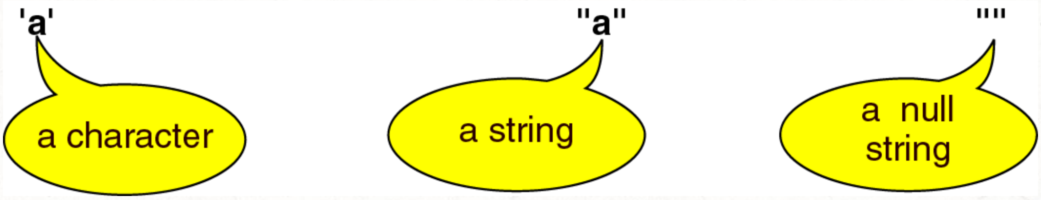
Strings



Character vs. Strings

- A string constant is a sequence of characters enclosed in double quotes.
  - For example, the character string:
  - `char s1[2]="a"; //Takes two bytes of storage.`
  - s1: a \0
  - On the other hand, the character, in single quotes:
  - `char s2= `a`;//Takes only one byte of storage.`
  - s2: a

Character vs. Strings



50

```
void printCharArray(const char[],int);

int main()
{
            char first = 'C';
            char second = 'P';

            cout << first << second << endl;

            char courseTitle1[8] = { 'C','o', 'm', 'p', 'u', 't','e','r' };
            char courseTitle2[12] = { 'P','r', 'o', 'g', 'r', 'a','m','m','i','n','g','\0' };
            char myName[] = "Muhammad Izaan Ali";

            char courseTitle[] = "Computer Programming 1B";
            const char* topicForToday = "We are uderstanding strings";

            printCharArray(topicForToday, strlen(topicForToday));
            cout << topicForToday << endl;
            cout << courseTitle << endl;

            printCharArray(courseTitle1,8);
            cout << courseTitle2 << endl;

            //cout << myName << endl;

            printCharArray(myName, 19);

            _getch();
            }

            void printCharArray(const char charArray[],int size)
            {
                  for (int index = 0; index < size; index++)
                  {
                  cout << charArray[index];
                  }
                  }
}
```

51

Strings – Example

```
char message1[12] = "Hello world";
cout << message1 << endl;
```

message1:

H	e	l	l	o		w	o	r	l	d	\0

```
char message2[12];
cin >> message2;   // type "Hello" as input
```

message2:

H	e	l	l	o	\0	?	?	?	?	?	?

Fundamentals of Characters and Strings

- String assignment
  - Character array
    - `char color[] = "blue";`
      - Creates 5 element `char` array `color`
        - last element is `'\0'`
  - Variable of type `char *`
    - `char *colorPtr = "blue";`
      - Creates pointer `colorPtr` to letter `b` in string “`blue`”
        - “`blue`” somewhere in memory
  - Alternative for character array
    - `char color[] = { ‘b’, ‘l’, ‘u’, ‘e’, ‘\0’ };`

Fundamentals of Characters and Strings

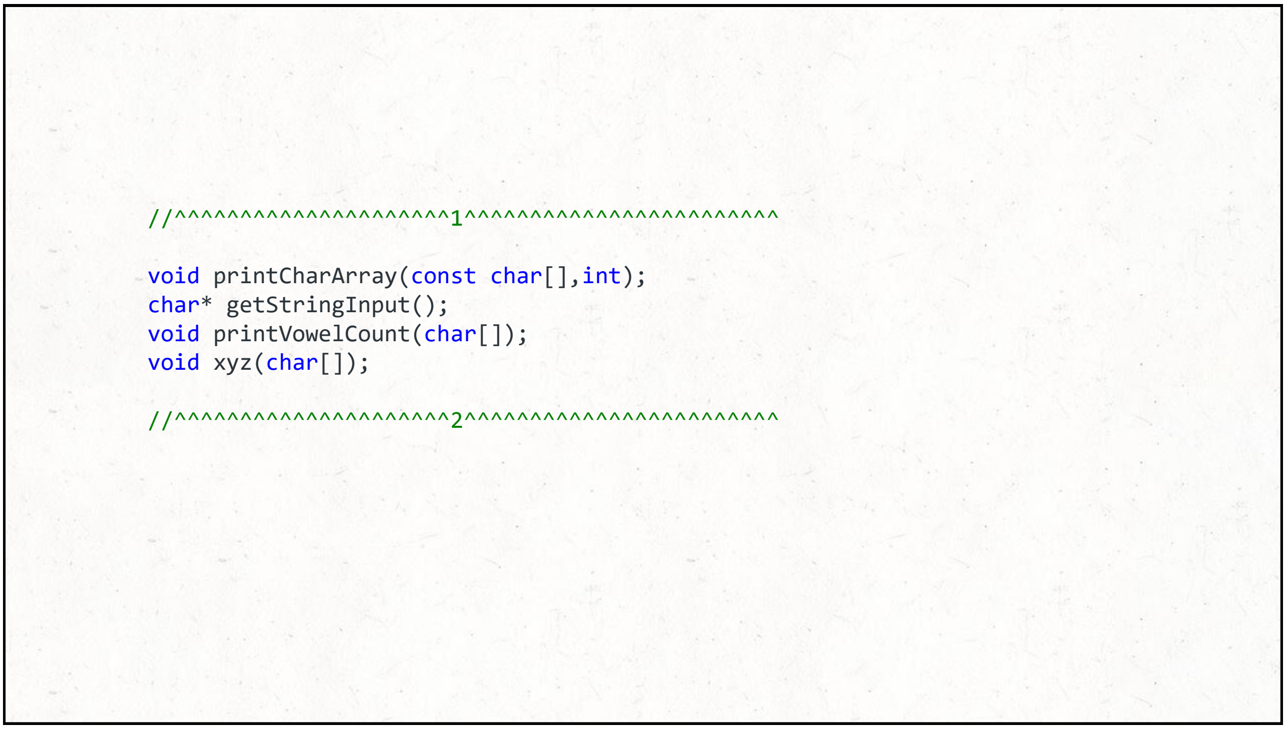
- Reading strings
  - ○ Assign input to character array **`word[ 20 ]`**

    ```
    cin >> word
    ```
    - ■ Reads characters until whitespace
    - ■ Reads 19 characters (space reserved for **`'\0'`**)

Fundamentals of Characters and Strings

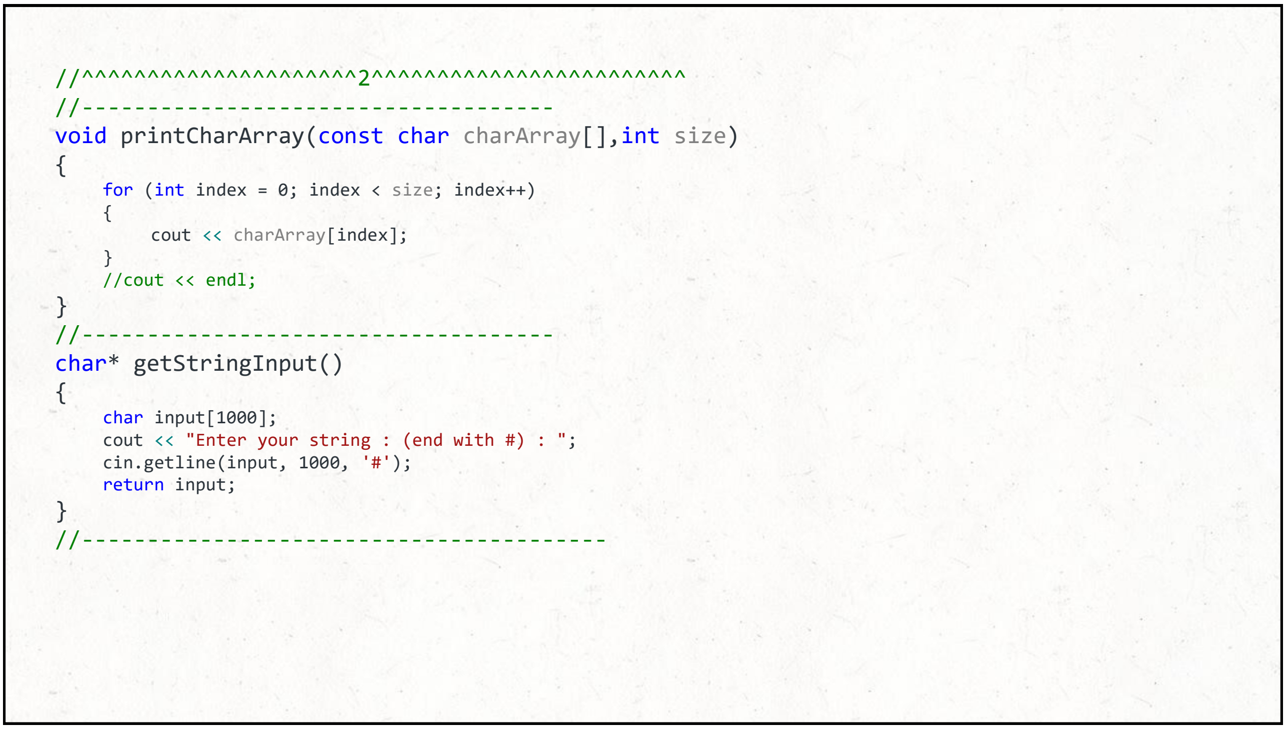
- **`cin.getline`**
  - ○ Read line of text
  - ○ **`cin.getline( array, size, delimiter );`**
  - ○ Copies input into specified **`array`** until either
    - ■ One less than **`size`** is reached
    - ■ **`delimiter`** character is input
  - ○ Example

    ```
    char sentence[ 80 ];
    cin.getline( sentence, 80, '\n' );
    ```



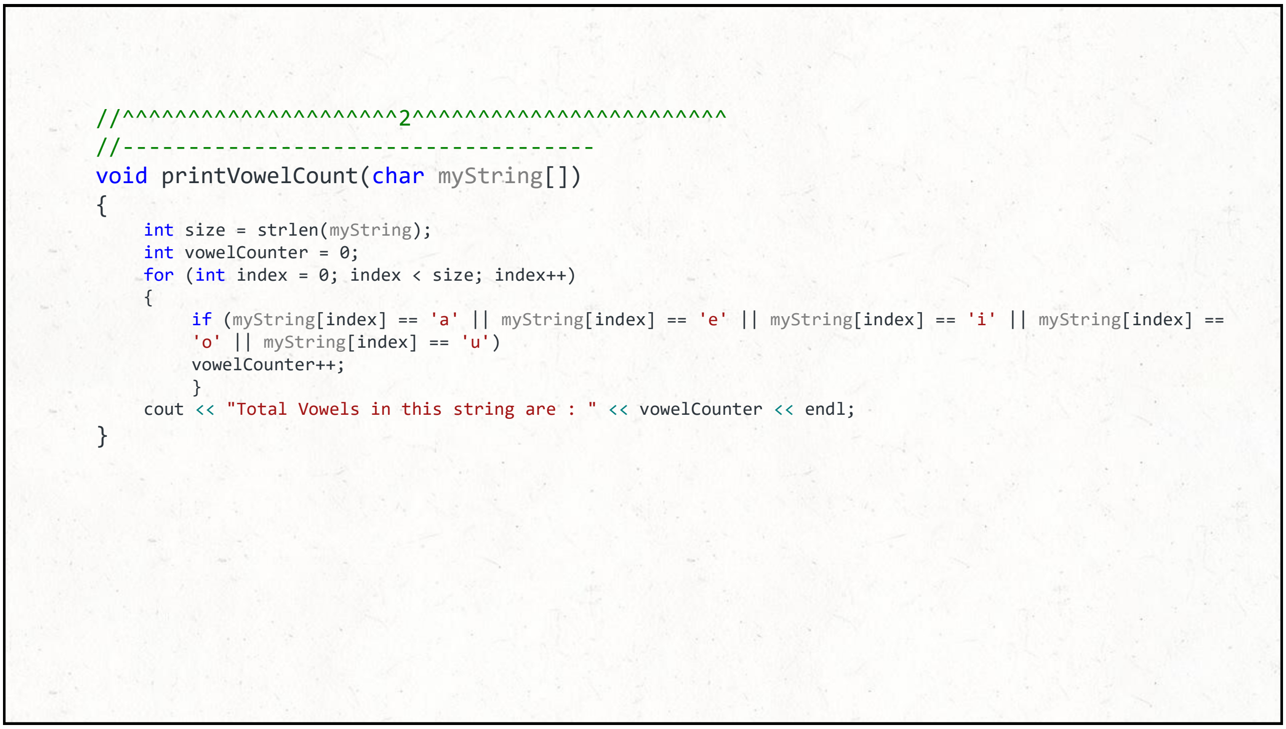
```
//^^^^^^^^^^^^^^^^^^^^^1^^^^^^^^^^^^^^^^^^^^^^^^

void printCharArray(const char[],int);
char* getStringInput();
void printVowelCount(char[]);
void xyz(char[]);

//^^^^^^^^^^^^^^^^^^^^^2^^^^^^^^^^^^^^^^^^^^^^^^
```

56



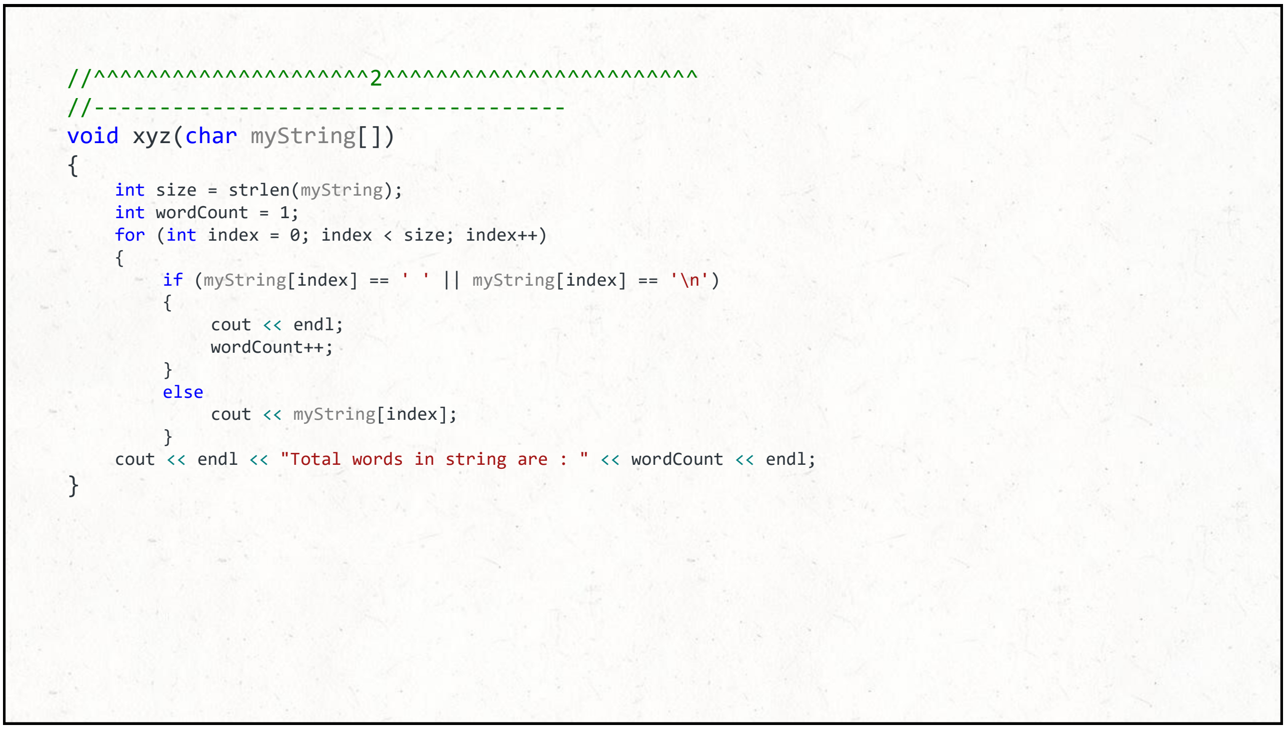
```
//^^^^^^^^^^^^^^^^^^^^^2^^^^^^^^^^^^^^^^^^^^^^^^
//------------------------------------
void printCharArray(const char charArray[],int size)
{
    for (int index = 0; index < size; index++)
    {
        cout << charArray[index];
    }
    //cout << endl;
}
//------------------------------------
char* getStringInput()
{
    char input[1000];
    cout << "Enter your string : (end with #) : ";
    cin.getline(input, 1000, '#');
    return input;
}
//----------------------------------------
```

57



```
//^^^^^^^^^^^^^^^^^^^^^2^^^^^^^^^^^^^^^^^^^^^^^^
//-----------------------------------
void printVowelCount(char myString[])
{
    int size = strlen(myString);
    int vowelCounter = 0;
    for (int index = 0; index < size; index++)
    {
        if (myString[index] == 'a' || myString[index] == 'e' || myString[index] == 'i' || myString[index] ==
        'o' || myString[index] == 'u')
        vowelCounter++;
        }
    cout << "Total Vowels in this string are : " << vowelCounter << endl;
}
```

58



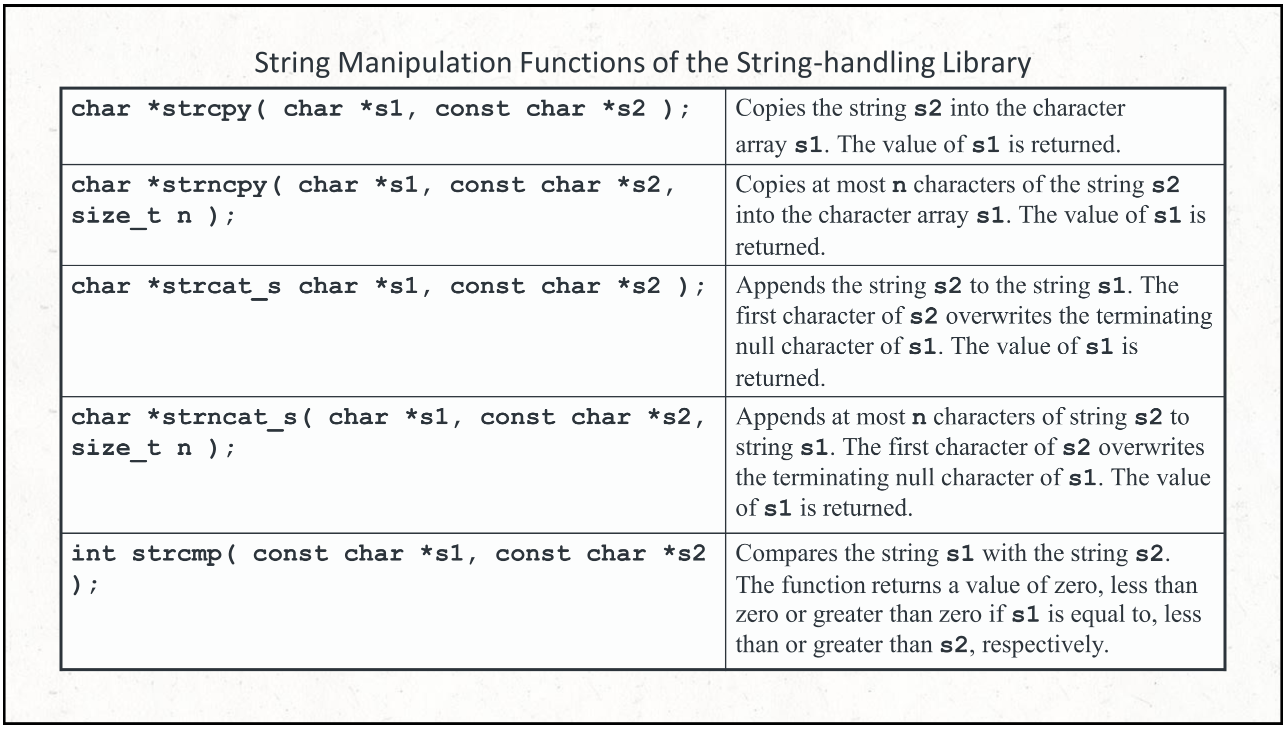
```
//^^^^^^^^^^^^^^^^^^^^^2^^^^^^^^^^^^^^^^^^^^^^^^
//-----------------------------------
void xyz(char myString[])
{
    int size = strlen(myString);
    int wordCount = 1;
    for (int index = 0; index < size; index++)
    {
        if (myString[index] == ' ' || myString[index] == '\n')
        {
            cout << endl;
            wordCount++;
        }
        else
            cout << myString[index];
        }
    cout << endl << "Total words in string are : " << wordCount << endl;
}
```

59

```
//---------------------------------------
int main()
{
    const int size = 1000;
    char courseTitle[size];
    cout << "Enter your course title : ";
    //cin >> courseTitle;
    cin.getline(courseTitle, size, '^');
    cout << "---------------------------" << endl;
    cout << "Welcome to " << courseTitle << endl;

    //char* myInputString = new char[1000];
    //myInputString = getStringInput();
    //cout << myInputString << endl;

    printVowelCount(courseTitle);
    xyz(courseTitle);
    _getch();
}
//---------------------------------------
```

String Manipulation Functions of the String-handling Library

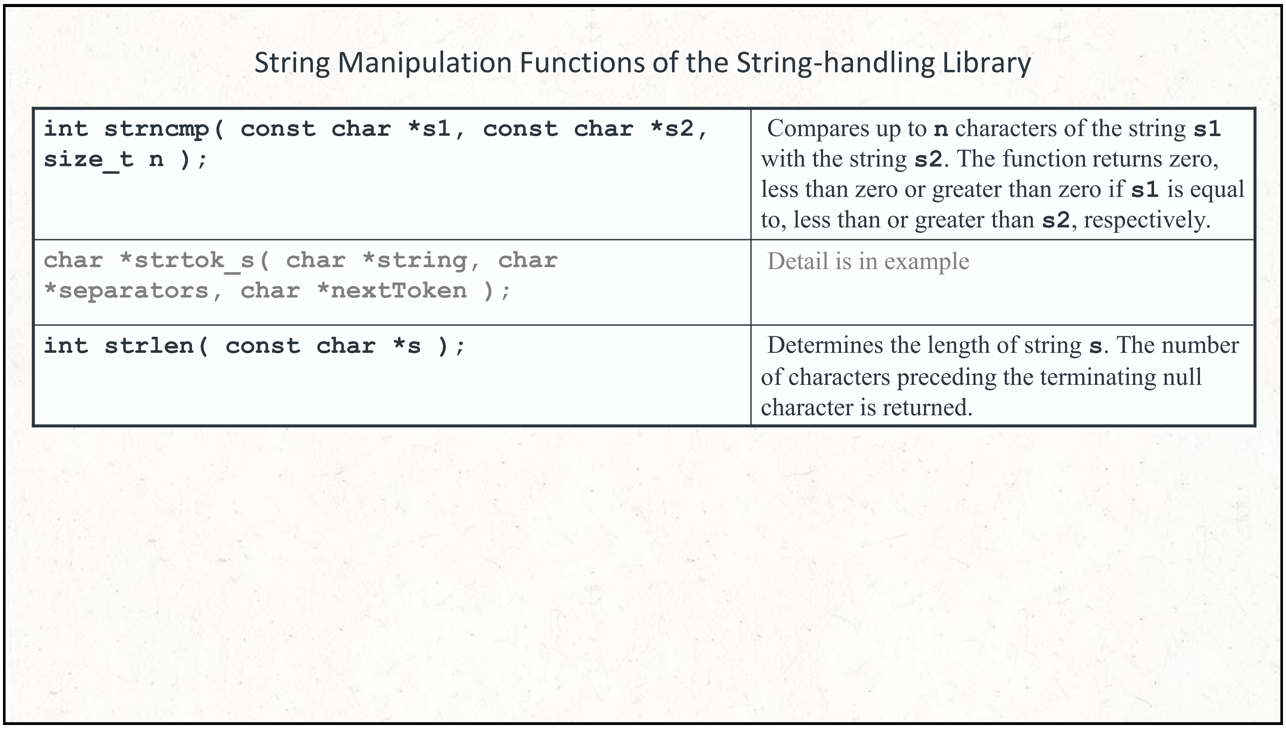
- String handling library **`<cstring>`** provides functions to
  - Manipulate string data
  - Compare strings
  - Search strings for characters and other strings
  - Tokenize strings (separate strings into logical pieces)



String Manipulation Functions of the String-handling Library

`char *strcpy( char *s1, const char *s2 );`	Copies the string **s2** into the character array **s1**. The value of **s1** is returned.
`char *strncpy( char *s1, const char *s2, size_t n );`	Copies at most **n** characters of the string **s2** into the character array **s1**. The value of **s1** is returned.
`char *strcat_s char *s1, const char *s2 );`	Appends the string **s2** to the string **s1**. The first character of **s2** overwrites the terminating null character of **s1**. The value of **s1** is returned.
`char *strncat_s( char *s1, const char *s2, size_t n );`	Appends at most **n** characters of string **s2** to string **s1**. The first character of **s2** overwrites the terminating null character of **s1**. The value of **s1** is returned.
`int strcmp( const char *s1, const char *s2 );`	Compares the string **s1** with the string **s2**. The function returns a value of zero, less than zero or greater than zero if **s1** is equal to, less than or greater than **s2**, respectively.

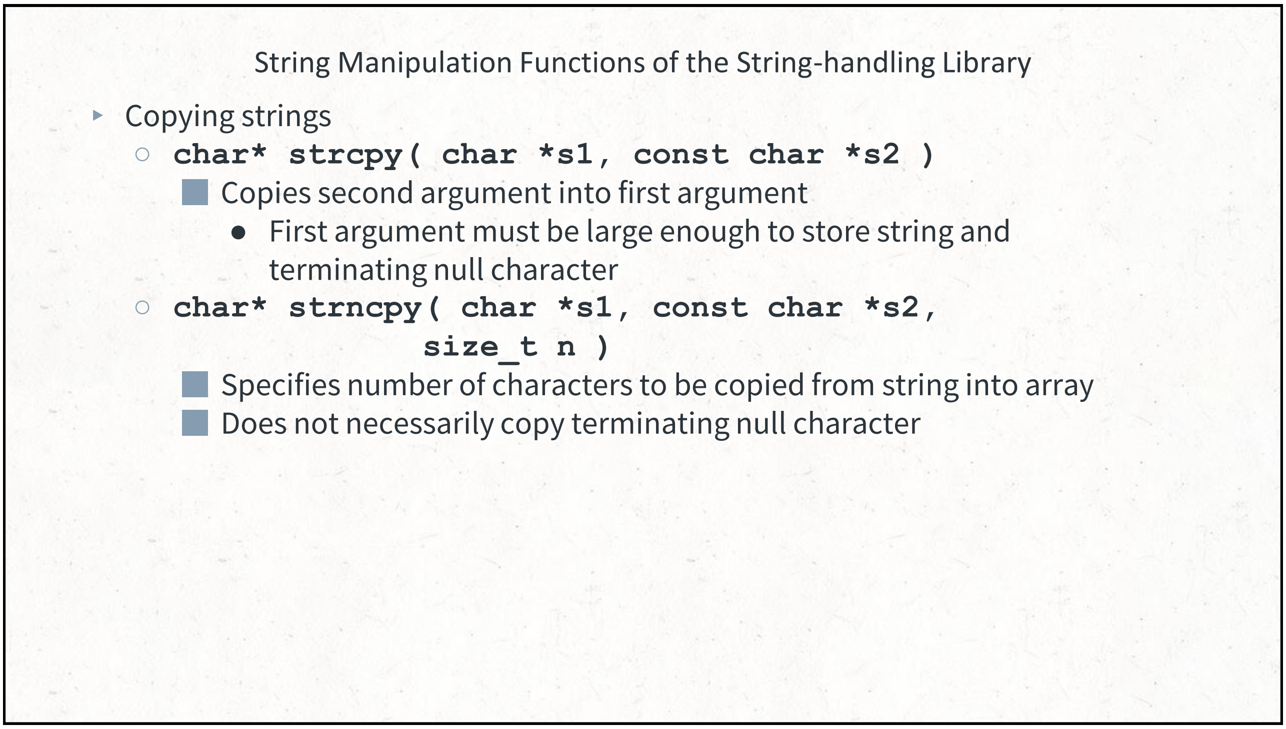
62



String Manipulation Functions of the String-handling Library

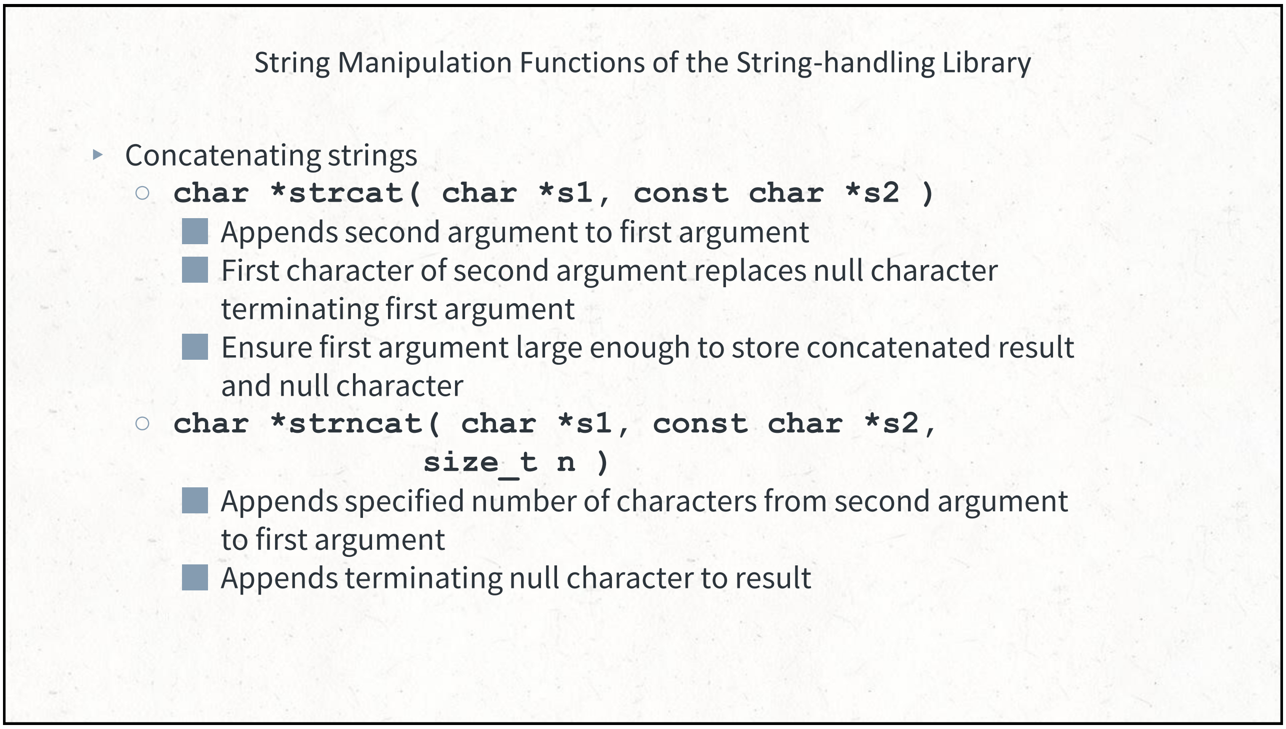
`int strncmp( const char *s1, const char *s2, size_t n );`	Compares up to **n** characters of the string **s1** with the string **s2**. The function returns zero, less than zero or greater than zero if **s1** is equal to, less than or greater than **s2**, respectively.
`char *strtok_s( char *string, char *separators, char *nextToken );`	Detail is in example
`int strlen( const char *s );`	Determines the length of string **s**. The number of characters preceding the terminating null character is returned.

63



String Manipulation Functions of the String-handling Library

- Copying strings
  - `char* strcpy( char *s1, const char *s2 )`
    - Copies second argument into first argument
      - First argument must be large enough to store string and terminating null character
  - `char* strncpy( char *s1, const char *s2, size_t n )`
    - Specifies number of characters to be copied from string into array
    - Does not necessarily copy terminating null character

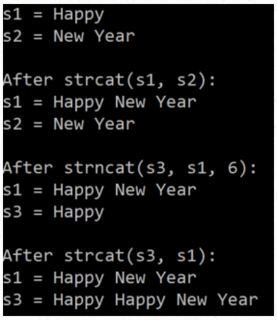


String Manipulation Functions of the String-handling Library

- Concatenating strings
  - `char *strcat( char *s1, const char *s2 )`
    - Appends second argument to first argument
    - First character of second argument replaces null character terminating first argument
    - Ensure first argument large enough to store concatenated result and null character
  - `char *strncat( char *s1, const char *s2, size_t n )`
    - Appends specified number of characters from second argument to first argument
    - Appends terminating null character to result

Example

```
void main()
{
    char s1[20] = "Happy ";
    char s2[] = "New Year ";
    char s3[40] = "";
    cout << "s1 = " << s1 << "\ns2 = " << s2;

    strcat_s(s1, s2);  // concatenate s2 to s1

    cout << "\n\nAfter strcat_s(s1, s2):\ns1 = " << s1 << "\ns2 = " << s2;

    // concatenate first 6 characters of s1 to s3
    strncat_s(s3, s1, 6);  // places '\0' after last character
    cout << "\n\nAfter strncat_s(s3, s1, 6):\ns1 = " << s1 << "\ns3 = " << s3;

    strcat_s(s3, s1);  // concatenate s1 to s3
    cout << "\n\nAfter strcat_s(s3, s1):\ns1 = " << s1 << "\ns3 = " << s3 << endl;
    _getch();
}
```



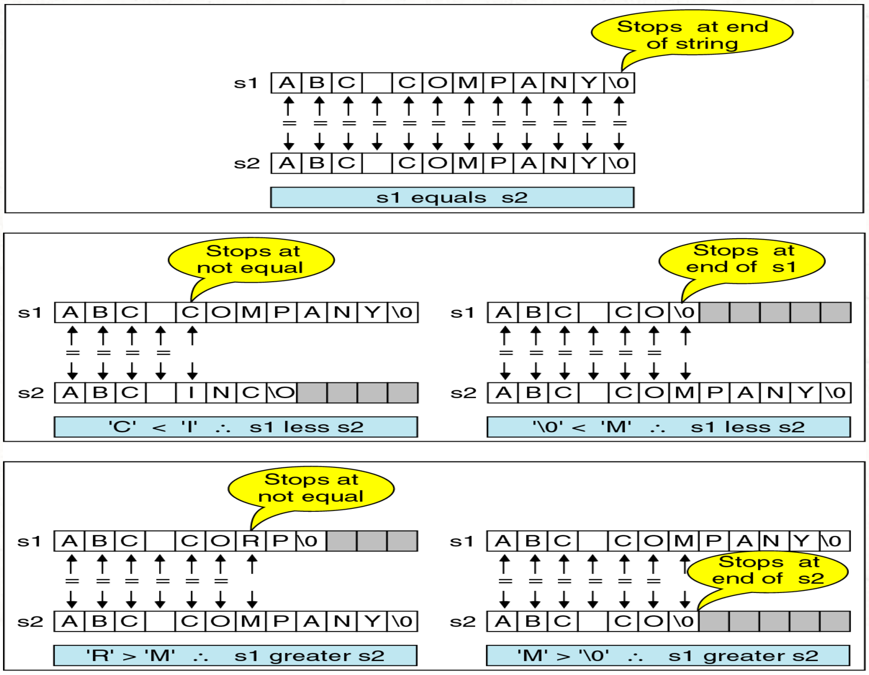
```
s1 = Happy
s2 = New Year

After strcat(s1, s2):
s1 = Happy New Year
s2 = New Year

After strncat(s3, s1, 6):
s1 = Happy New Year
s3 = Happy

After strcat(s3, s1):
s1 = Happy New Year
s3 = Happy Happy New Year
```

String Manipulation Functions of the String-handling Library

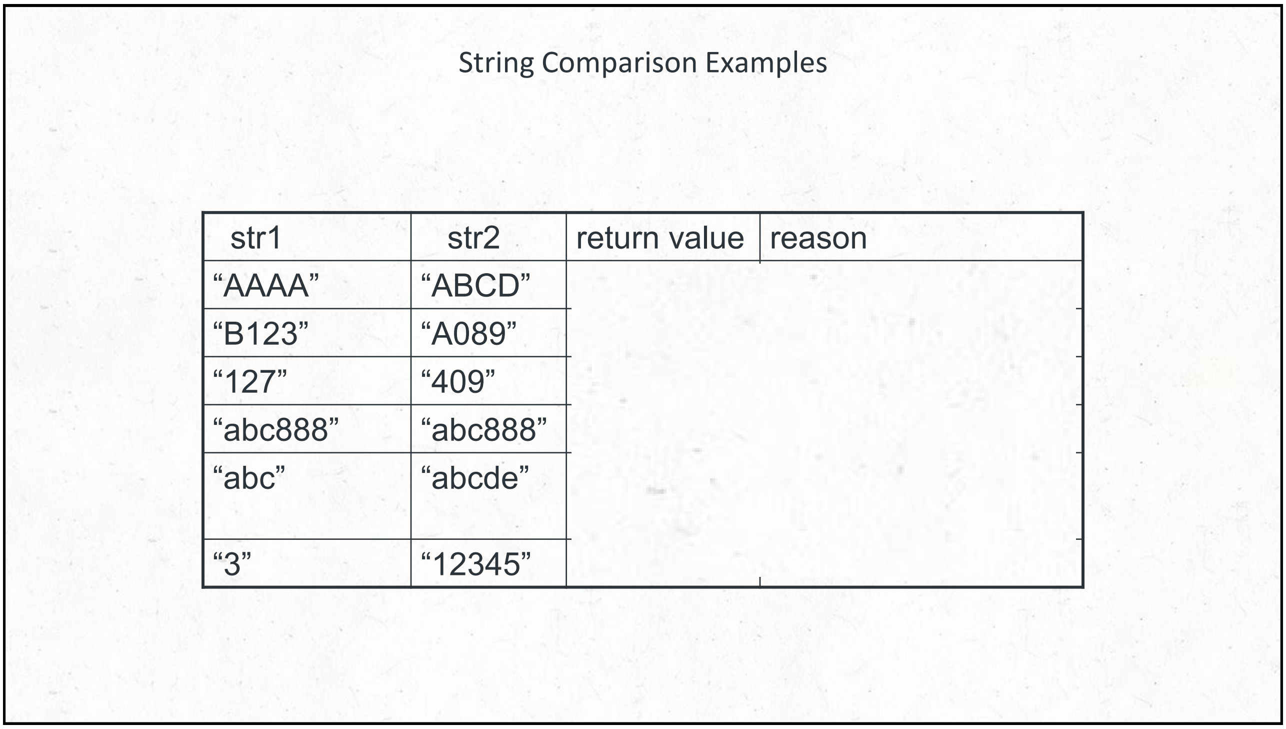
- Comparing strings
    - Characters represented as numeric codes
        - Strings compared using numeric codes
    - Character codes / character sets
        - ASCII
            - “American Standard Code for Information Interchage”
        - EBCDIC
            - “Extended Binary Coded Decimal Interchange Code”

String Manipulation Functions of the String-handling Library

- Comparing strings
  - `int strcmp( const char *s1, const char *s2 )`
    - Compares character by character
    - Returns
      - Zero if strings equal
      - Negative value if first string less than second string
      - Positive value if first string greater than second string
  - `int strncmp( const char *s1,`
    `const char *s2, size_t n )`
    - Compares up to specified number of characters
    - Stops comparing if reaches null character in one of arguments



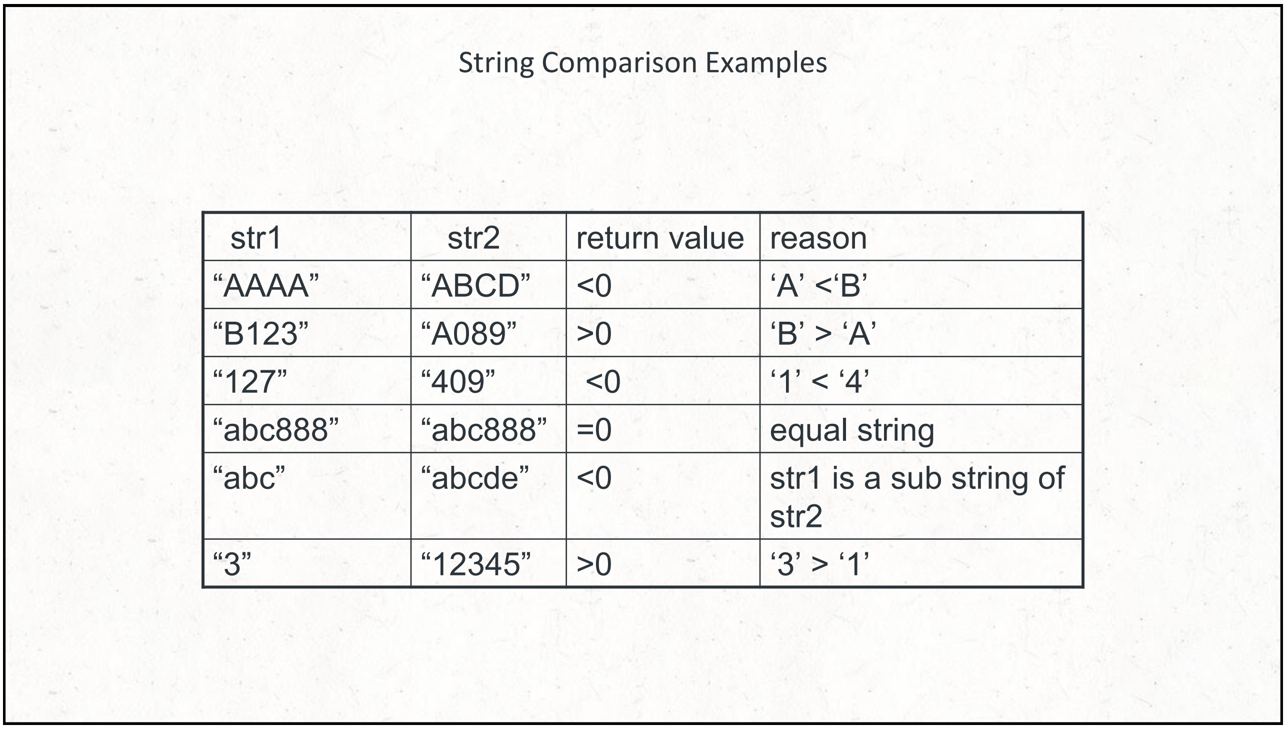
strcmp (s1, s2)



String Comparison Examples

str1	str2	return value	reason
“AAAA”	“ABCD”		
“B123”	“A089”		
“127”	“409”		
“abc888”	“abc888”		
“abc”	“abcde”		
“3”	“12345”		

70



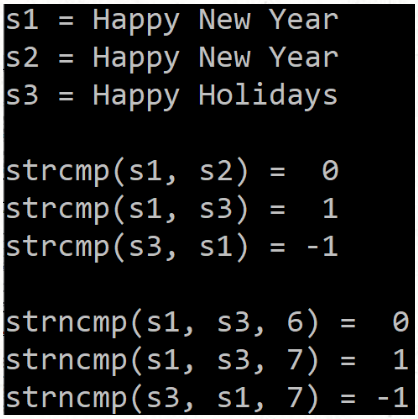
String Comparison Examples

str1	str2	return value	reason
“AAAA”	“ABCD”	<0	‘A’ <‘B’
“B123”	“A089”	>0	‘B’ > ‘A’
“127”	“409”	<0	‘1’ < ‘4’
“abc888”	“abc888”	=0	equal string
“abc”	“abcde”	<0	str1 is a sub string of str2
“3”	“12345”	>0	‘3’ > ‘1’

71

Example

```cpp
void main()
{
     const char* s1 = "Happy New Year";
     const char* s2 = "Happy New Year";
     const char* s3 = "Happy Holidays";

     cout << "s1 = " << s1 << "\ns2 = " << s2
             << "\ns3 = " << s3 << "\n\nstrcmp(s1, s2) = "
             << setw(2) << strcmp(s1, s2)
             << "\nstrcmp(s1, s3) = " << setw(2)
             << strcmp(s1, s3) << "\nstrcmp(s3, s1) = "
             << setw(2) << strcmp(s3, s1);

     cout << "\n\nstrncmp(s1, s3, 6) = " << setw(2)
             << strncmp(s1, s3, 6) << "\nstrncmp(s1, s3, 7) = "
             << setw(2) << strncmp(s1, s3, 7)
             << "\nstrncmp(s3, s1, 7) = "
             << setw(2) << strncmp(s3, s1, 7) << endl;
}
```



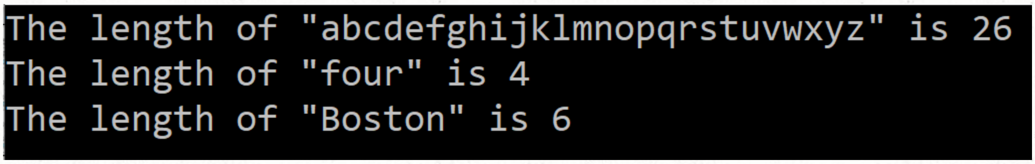
```
s1 = Happy New Year
s2 = Happy New Year
s3 = Happy Holidays

strcmp(s1, s2) =  0
strcmp(s1, s3) =  1
strcmp(s3, s1) = -1

strncmp(s1, s3, 6) =  0
strncmp(s1, s3, 7) =  1
strncmp(s3, s1, 7) = -1
```

String Manipulation Functions of the String-handling Library

- Determining string lengths
  - **`int strlen( const char *s )`**
    - Returns number of characters in string
      - Terminating null character not included in length

Example

```
void main()
{
     const char* string1 = "abcdefghijklmnopqrstuvwxyz";
     const char* string2 = "four";
     const char* string3 = "Boston";

     cout << "The length of \"" << string1
               << "\" is " << strlen(string1)
               << "\nThe length of \"" << string2
               << "\" is " << strlen(string2)
               << "\nThe length of \"" << string3
               << "\" is " << strlen(string3) << endl;
}
```



```
The length of "abcdefghijklmnopqrstuvwxyz" is 26
The length of "four" is 4
The length of "Boston" is 6
```

Some Common Errors

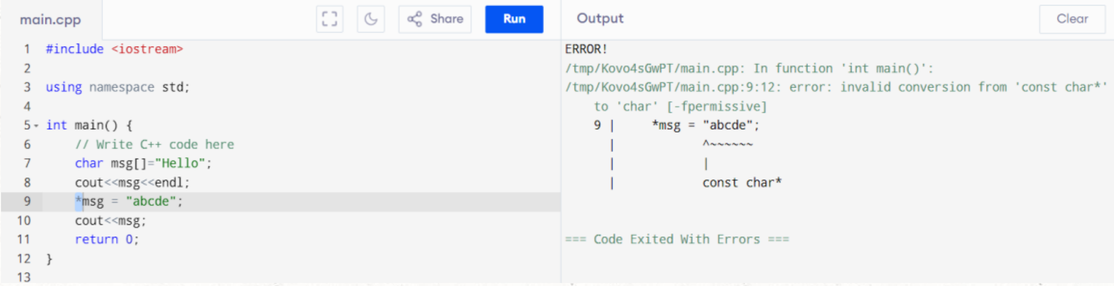
It is illegal to assign a value to a string variable (except at declaration).

```
char A_string[10];
A_string = "Hello‟; // illegal
```

Should use instead

```
strcpy (A_string, "Hello");
```

Fundamentals of Characters and Strings



```cpp
#include <iostream>

using namespace std;

int main() {
    // Write C++ code here
    char msg[]="Hello";
    cout<<msg<<endl;
    *msg = "abcde";
    cout<<msg;
    return 0;
}

```

```
ERROR!
/tmp/Kovo4sGwPT/main.cpp: In function 'int main()':
/tmp/Kovo4sGwPT/main.cpp:9:12: error: invalid conversion from 'const char*'
    to 'char' [-fpermissive]
    9 |     *msg = "abcde";
      |            ^~~~~~~
      |            |
      |            const char*


=== Code Exited With Errors ===
```

The operator == doesn't test two strings for equality.

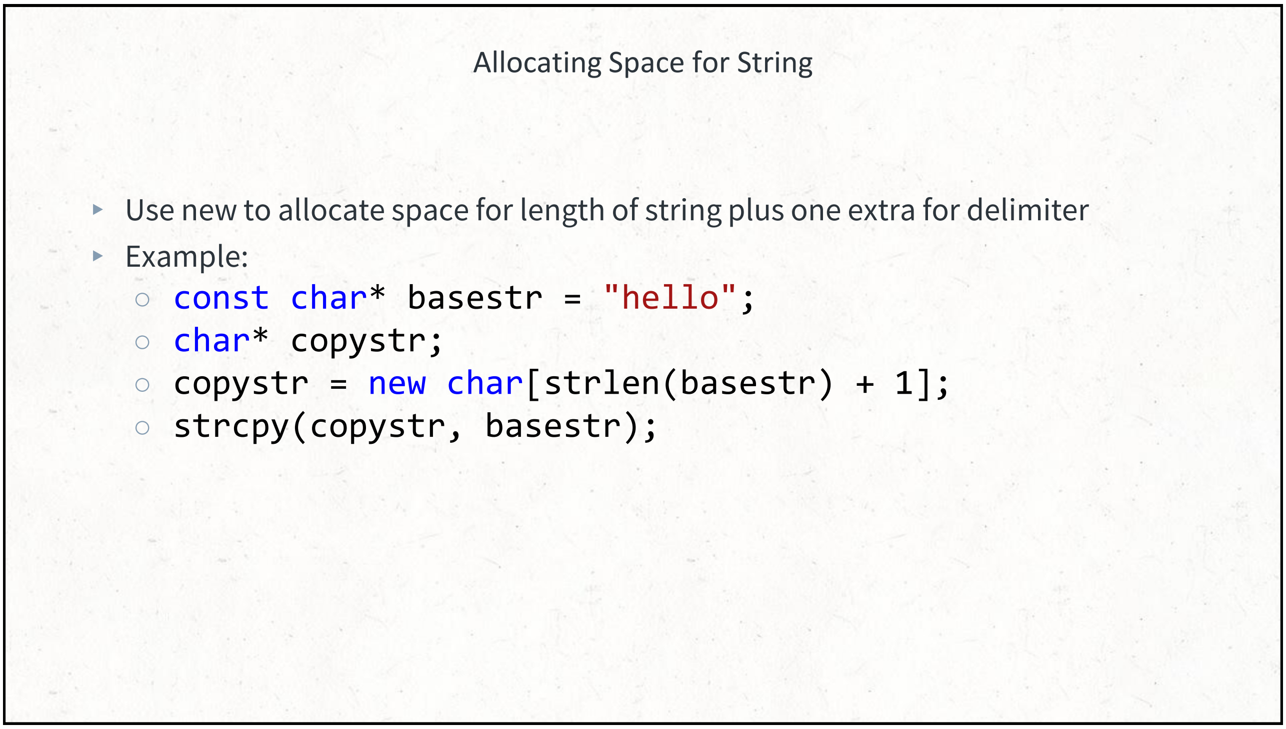
```
if (string1 == string2) //wrong
     cout << "Yes!";
```

Outline

Some Common Errors

Should use instead

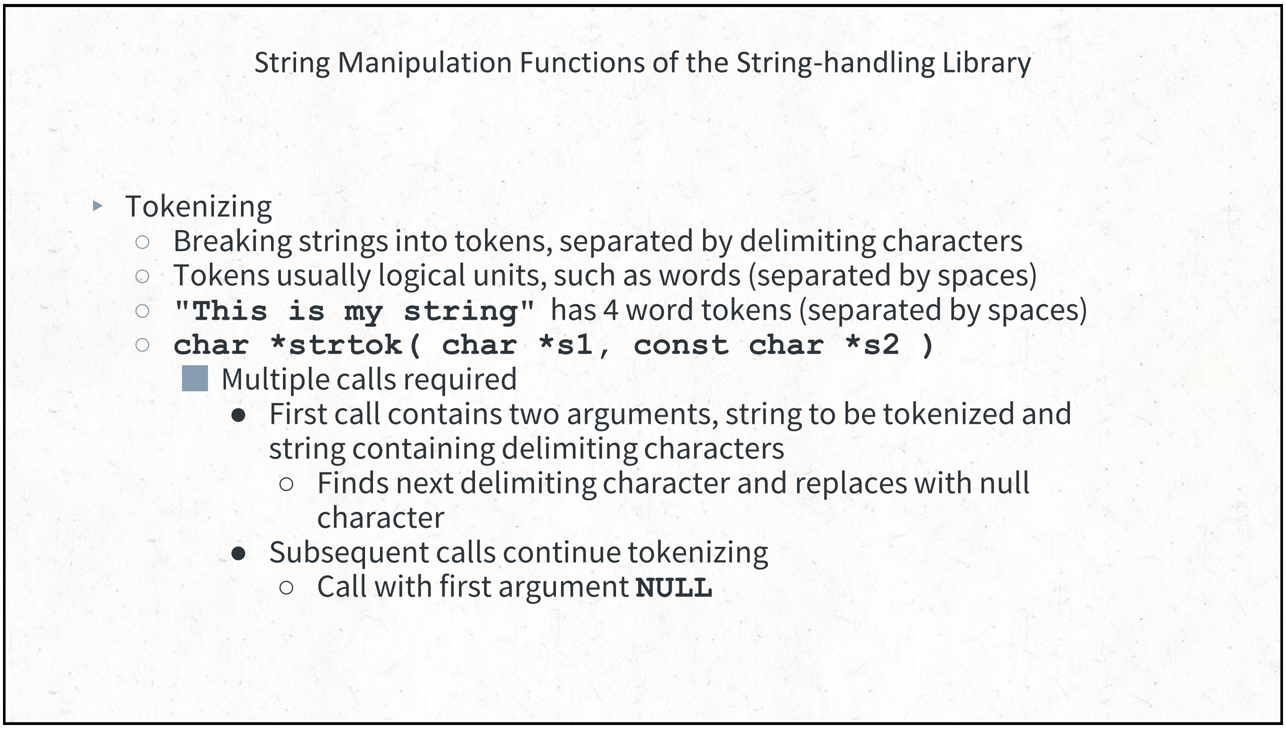
```
if (!strcmp(string1,string2))
     cout << "Yes they are same!";
```



Allocating Space for String

- Use new to allocate space for length of string plus one extra for delimiter
- Example:
  - `const char* basestr = "hello";`
  - `char* copystr;`
  - `copystr = new char[strlen(basestr) + 1];`
  - `strcpy(copystr, basestr);`

78



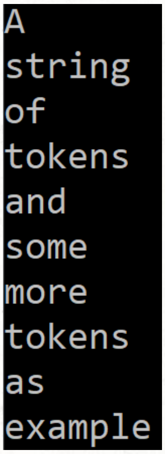
String Manipulation Functions of the String-handling Library

- Tokenizing
  - Breaking strings into tokens, separated by delimiting characters
  - Tokens usually logical units, such as words (separated by spaces)
  - **`"This is my string"`** has 4 word tokens (separated by spaces)
  - **`char *strtok( char *s1, const char *s2 )`**
    - Multiple calls required
      - First call contains two arguments, string to be tokenized and string containing delimiting characters
        - Finds next delimiting character and replaces with null character
      - Subsequent calls continue tokenizing
        - Call with first argument **`NULL`**

79

Example

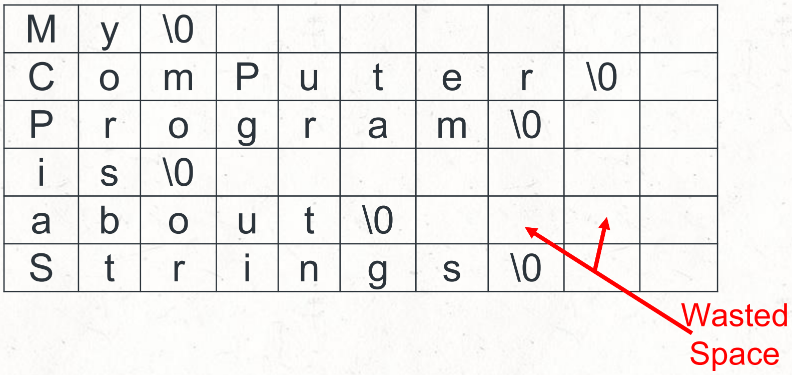
```
void main()
{
     char string[] = "A string\tof ,,tokens\nand some more tokens as example";
     char separators[] = " ,\t\n";
     char* token;
     char* next_token;

     // establish a string and get the first token:
     token = strtok_s(string, separators, &next_token);

     // while there are tokens in "string1" or "string2"
     while ((token != NULL))
     {
          // get the next token:
          if (token != NULL)
          {
               cout << token << endl;
               token = strtok_s(NULL, separators, &next_token);
          }
     }
}
```



```
A
string
of
tokens
and
some
more
tokens
as
example
```

Array of Strings

```
char listOfStrings[5][10] = {“My", "Computer", "Program", “is", “about", “Strings" };

for (int index = 0; index < 5; index++)
{
        cout << listOfStrings[index] << " ";
}
```



M	y	\0							
C	o	m	P	u	t	e	r	\0	
P	r	o	g	r	a	m	\0		
i	s	\0							
a	b	o	u	t	\0				
S	t	r	i	n	g	s	\0		

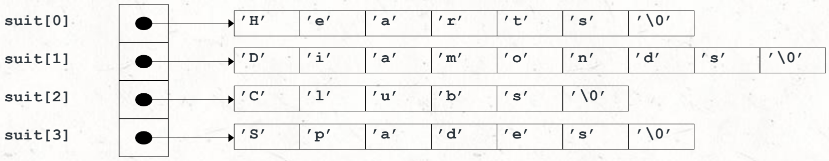
Wasted Space

Arrays of Pointers

- Arrays can contain pointers
  - Commonly used to store array of strings

```
char* suit[4] = { "Hearts", "Diamonds", "Clubs", "Spades" };
```

  - Each element of **suit** points to **`char *`** (a string)
  - Array does not store strings, only pointers to strings



suit[0] → 'H' 'e' 'a' 'r' 't' 's' '\0'

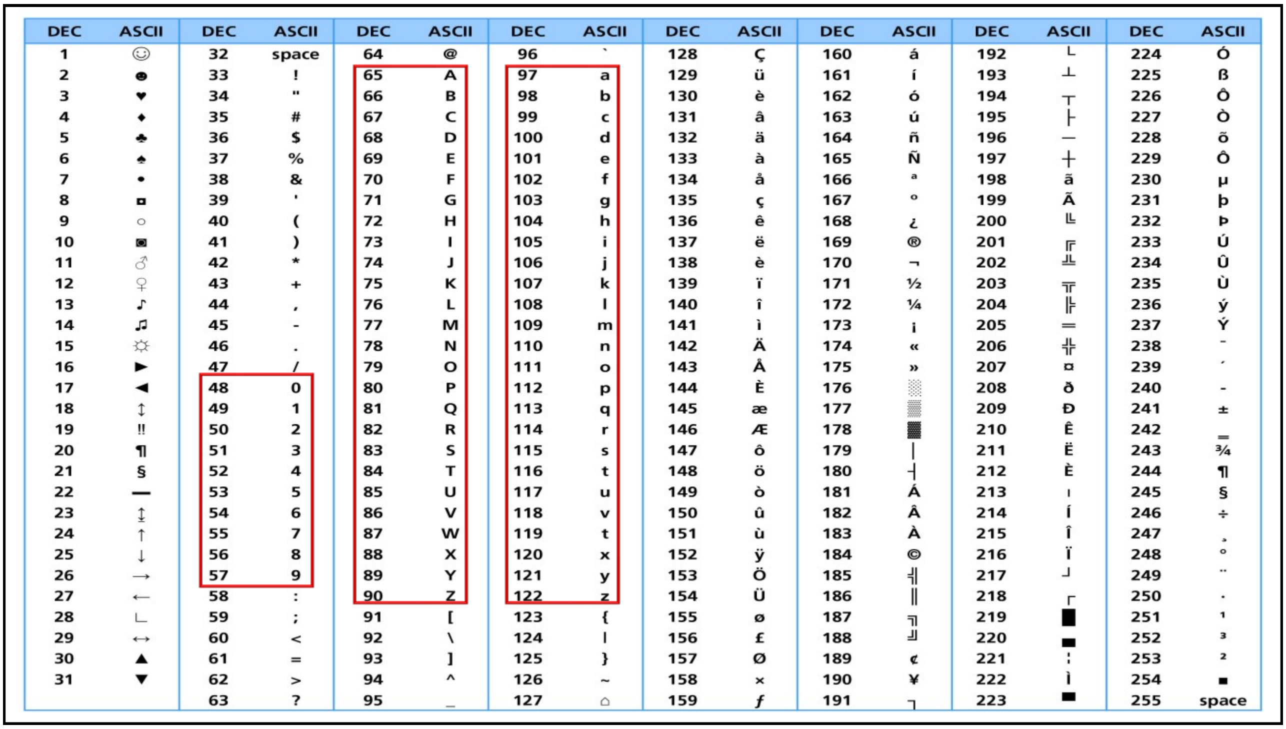
suit[1] → 'D' 'i' 'a' 'm' 'o' 'n' 'd' 's' '\0'

suit[2] → 'C' 'l' 'u' 'b' 's' '\0'

suit[3] → 'S' 'p' 'a' 'd' 'e' 's' '\0'

  - **`suit`** array has fixed size, but strings can be of any size

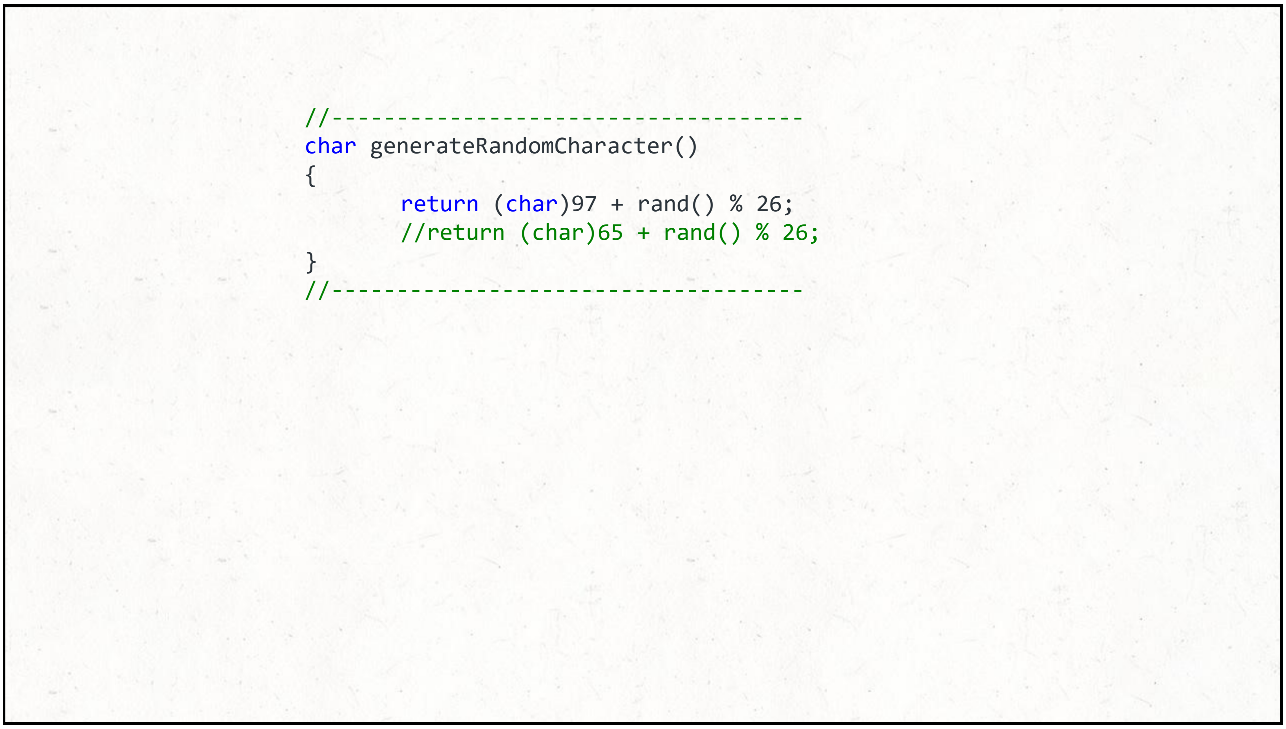
Interesting Example



DEC	ASCII	DEC	ASCII	DEC	ASCII	DEC	ASCII	DEC	ASCII	DEC	ASCII	DEC	ASCII	DEC	ASCII
1	☺	32	space	64	@	96	`	128	Ç	160	á	192	└	224	Ó
2	☻	33	!	65	A	97	a	129	ü	161	í	193	┴	225	ß
3	♥	34	"	66	B	98	b	130	è	162	ó	194	┬	226	Ô
4	♦	35	#	67	C	99	c	131	â	163	ú	195	├	227	Ò
5	♣	36	$	68	D	100	d	132	ä	164	ñ	196	─	228	õ
6	♠	37	%	69	E	101	e	133	à	165	Ñ	197	┼	229	Ô
7	•	38	&	70	F	102	f	134	å	166	ª	198	ã	230	µ
8	◘	39	'	71	G	103	g	135	ç	167	º	199	Ã	231	þ
9	○	40	(	72	H	104	h	136	ê	168	¿	200	╚	232	Þ
10	◙	41	)	73	I	105	i	137	ë	169	®	201	╔	233	Ú
11	♂	42	*	74	J	106	j	138	è	170	¬	202	╩	234	Û
12	♀	43	+	75	K	107	k	139	ï	171	½	203	╦	235	Ù
13	♪	44	,	76	L	108	l	140	î	172	¼	204	╠	236	ý
14	♫	45	-	77	M	109	m	141	ì	173	¡	205	═	237	Ý
15	☼	46	.	78	N	110	n	142	Ä	174	«	206	╬	238	¯
16	►	47	/	79	O	111	o	143	Å	175	»	207	¤	239	´
17	◄	48	0	80	P	112	p	144	È	176	░	208	ð	240	-
18	↕	49	1	81	Q	113	q	145	æ	177	▒	209	Ð	241	±
19	‼	50	2	82	R	114	r	146	Æ	178	▓	210	Ê	242	‗
20	¶	51	3	83	S	115	s	147	ô	179	│	211	Ë	243	¾
21	§	52	4	84	T	116	t	148	ö	180	┤	212	È	244	¶
22	▬	53	5	85	U	117	u	149	ò	181	Á	213	ı	245	§
23	↨	54	6	86	V	118	v	150	û	182	Â	214	Í	246	÷
24	↑	55	7	87	W	119	t	151	ù	183	À	215	Î	247	¸
25	↓	56	8	88	X	120	x	152	ÿ	184	©	216	Ï	248	°
26	→	57	9	89	Y	121	y	153	Ö	185	╣	217	┘	249	¨
27	←	58	:	90	Z	122	z	154	Ü	186	║	218	┌	250	·
28	∟	59	;	91	[	123	{	155	ø	187	╗	219	█	251	¹
29	↔	60	<	92	\	124	|	156	£	188	╝	220	▄	252	³
30	▲	61	=	93	]	125	}	157	Ø	189	¢	221	¦	253	²
31	▼	62	>	94	^	126	~	158	×	190	¥	222	Ì	254	■
		63	?	95	_	127	⌂	159	ƒ	191	┐	223	▀	255	space

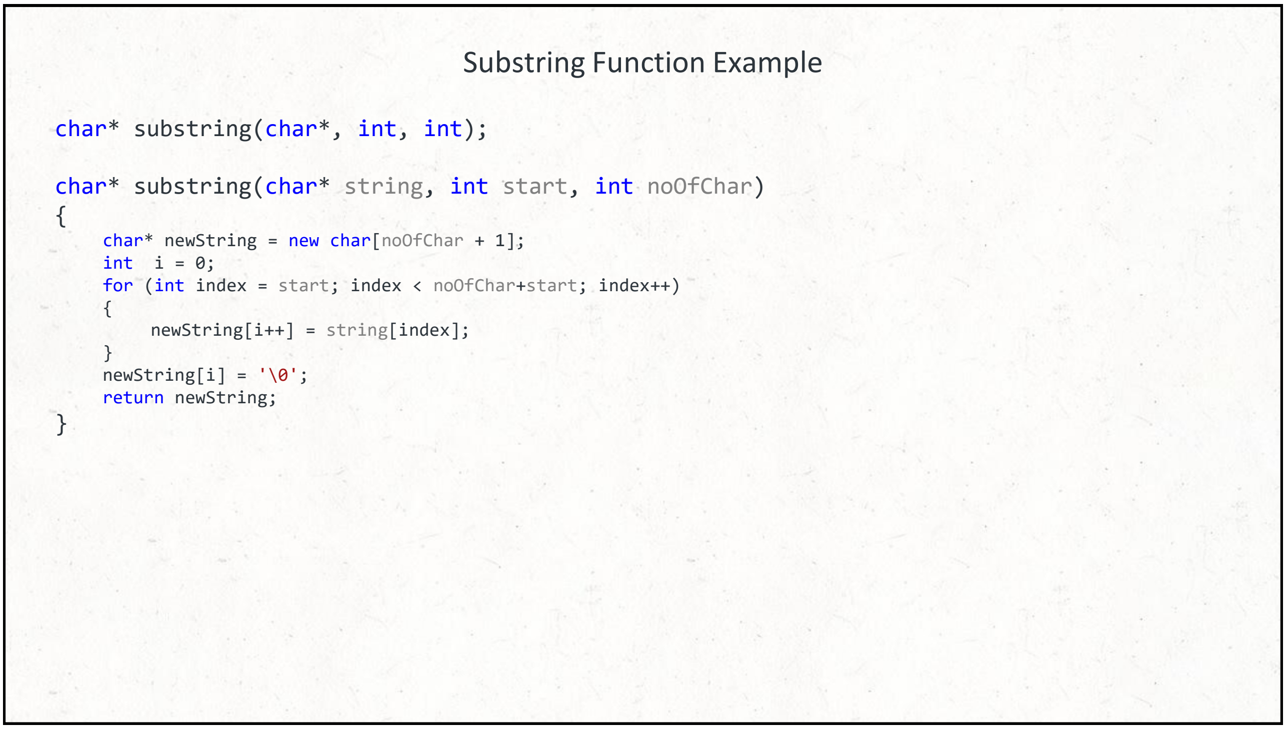
Recall Random numbers

```
//------------------------------------
char generateRandomNumber()
{
        return 1 + rand() % 10;

}
//------------------------------------
```



```
//-------------------------------------
char generateRandomCharacter()
{
        return (char)97 + rand() % 26;
        //return (char)65 + rand() % 26;
}
//-------------------------------------
```

86



Substring Function Example

```
char* substring(char*, int, int);

char* substring(char* string, int start, int noOfChar)
{
    char* newString = new char[noOfChar + 1];
    int  i = 0;
    for (int index = start; index < noOfChar+start; index++)
    {
        newString[i++] = string[index];
    }
    newString[i] = '\0';
    return newString;
}
```

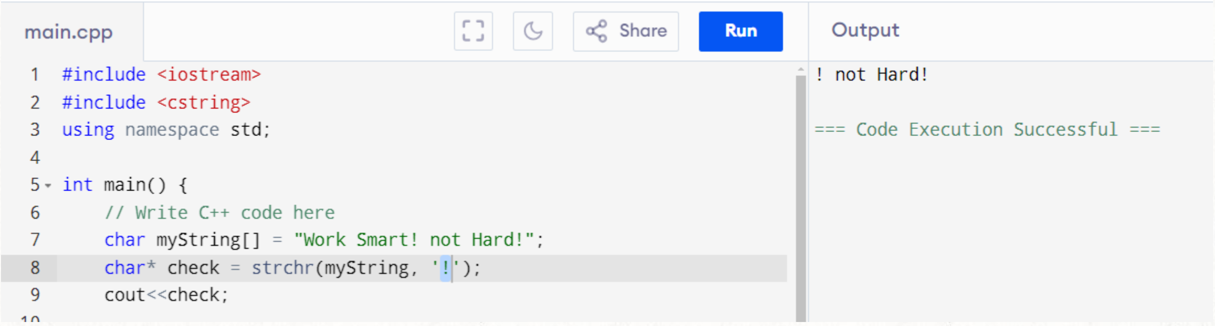
92

strchr

- It is used to find the first occurrence of a character in a C-style string
- If the character is found, it returns a pointer to that position in the string.
- If not found, it returns NULL.

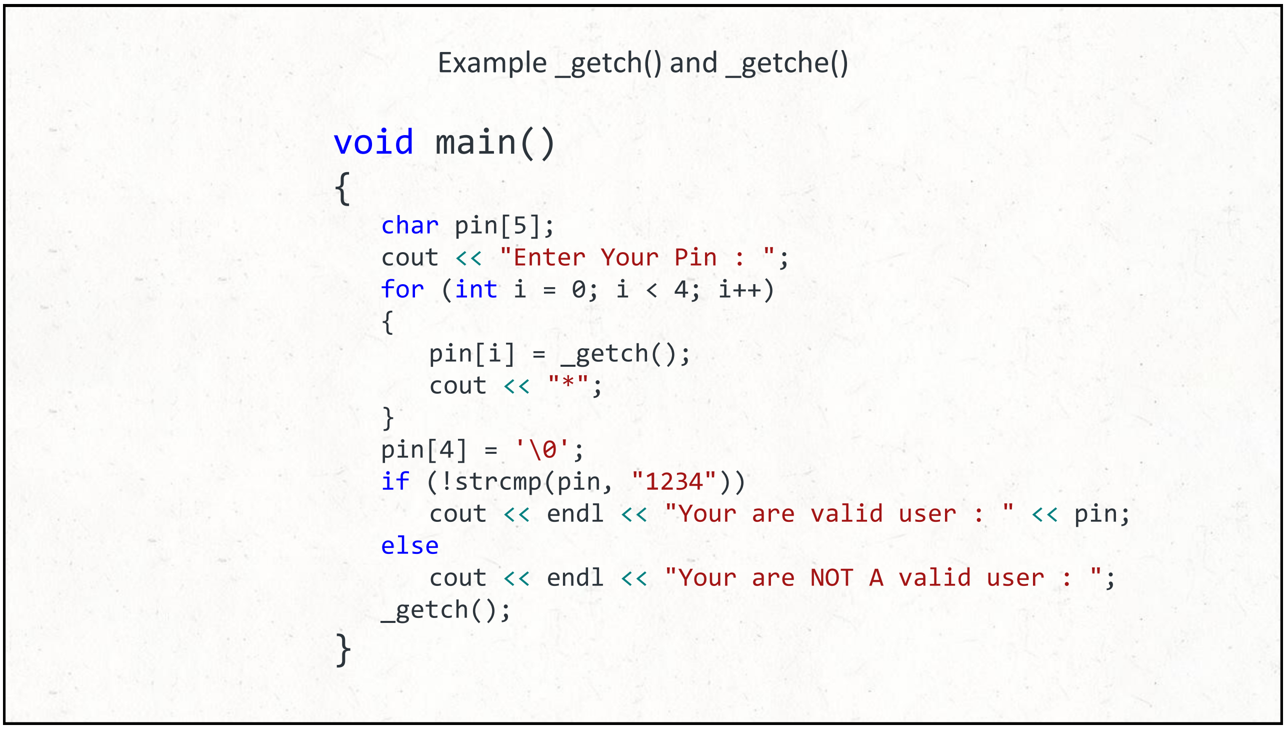
- Quick Question: What is a c-style string?
- character array terminated with '\0'.

strchr Exampl



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

int main() {
    // Write C++ code here
    char myString[] = "Work Smart! not Hard!";
    char* check = strchr(myString, '!');
    cout<<check;
```

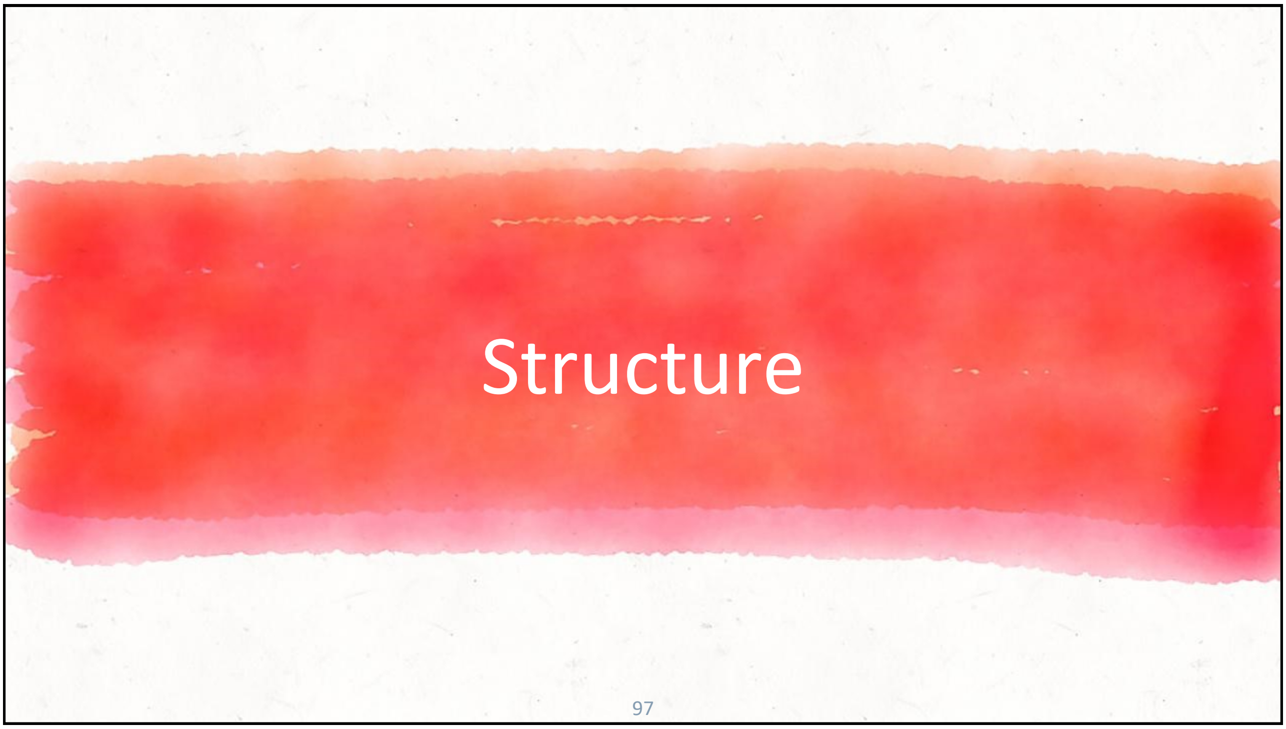
```
! not Hard!

=== Code Execution Successful ===
```



Example _getch() and _getche()

```
void main()
{
    char pin[5];
    cout << "Enter Your Pin : ";
    for (int i = 0; i < 4; i++)
    {
        pin[i] = _getch();
        cout << "*";
    }
    pin[4] = '\0';
    if (!strcmp(pin, "1234"))
        cout << endl << "Your are valid user : " << pin;
    else
        cout << endl << "Your are NOT A valid user : ";
    _getch();
}
```

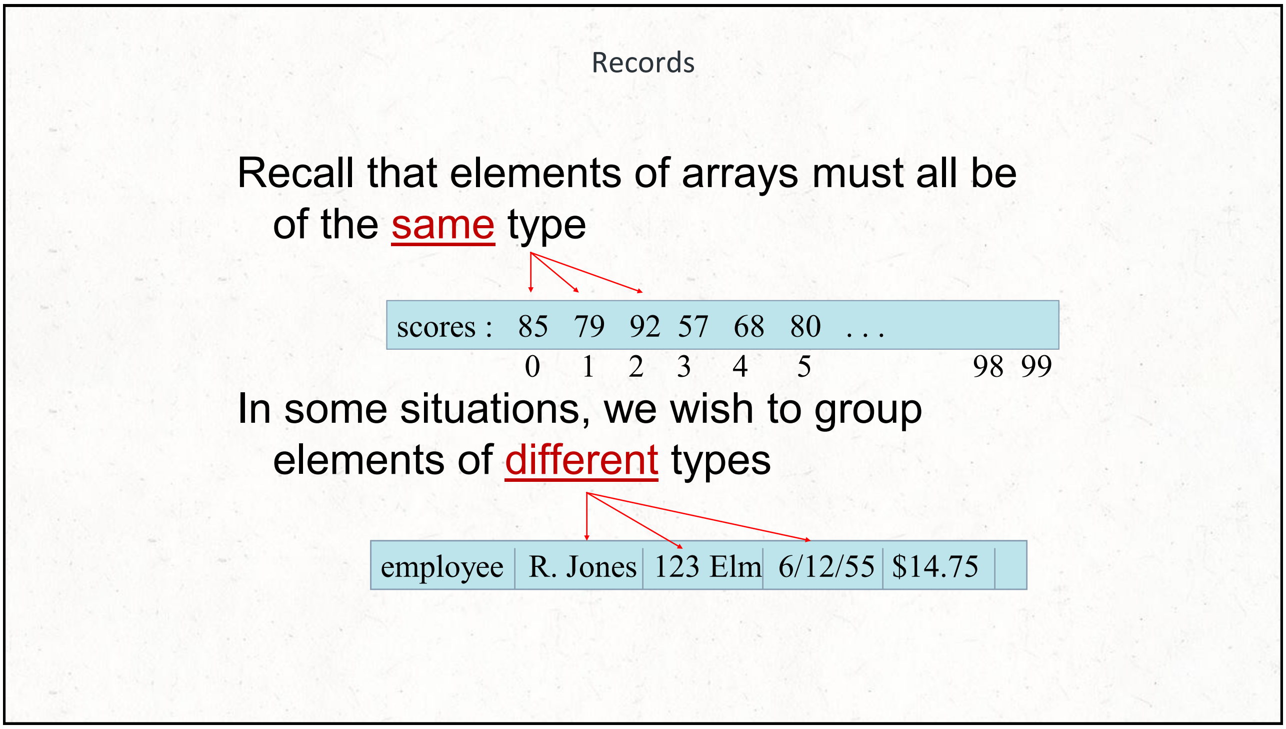
96



Structure

97

97



Records

Recall that elements of arrays must all be of the same type

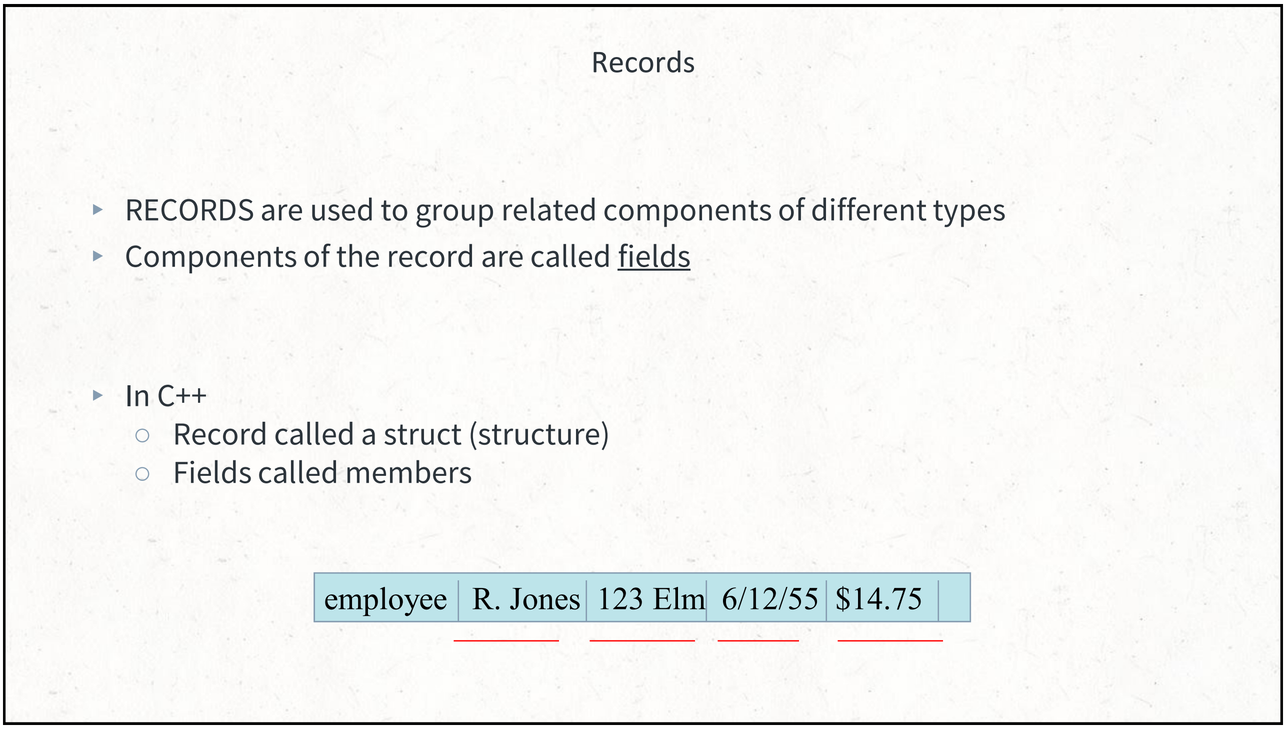
scores : 85 79 92 57 68 80 . . .

0 1 2 3 4 5 98 99

In some situations, we wish to group elements of different types

employee | R. Jones | 123 Elm | 6/12/55 | $14.75

98



Records

- RECORDS are used to group related components of different types
- Components of the record are called fields
- In C++
  - Record called a struct (structure)
  - Fields called members

employee | R. Jones | 123 Elm | 6/12/55 | $14.75

99

Structures

- A Structure is a collection of related data items, possibly of different types.
- A structure type in C++ is called struct.
- A struct is heterogeneous in that it can be composed of data of different types.
- In contrast, array is homogeneous since it can contain only data of the same type.

100

Structures

- Individual components of a struct type are called members (or fields).
- Members can be of different types (simple, array or struct).
- A struct is named as a whole while individual members are named using field identifiers.
- Complex data structures can be formed by defining arrays of structs.

101

Structures

- Structures hold data that belong **together**.
- Examples:
  - Student record: student id, name, major, gender, start year
  - Bank account: account number, name, currency, balance
  - Address book: name, address, telephone number
- In database applications, structures are called records.

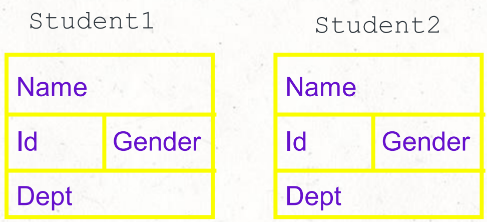
struct basics

- Declaration of a variable of struct type:

```
<struct-type> <identifier_list>;
```

- Example:

```
StudentRecord Student1, Student2;
```



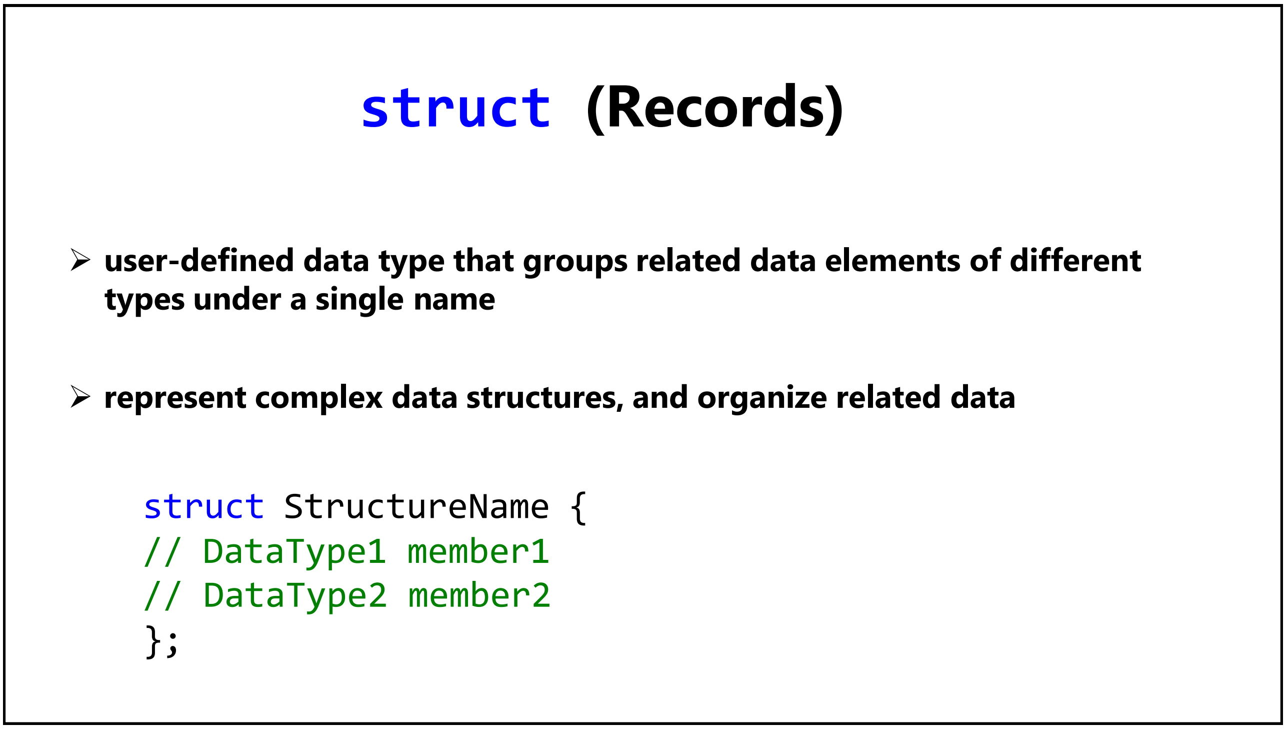
Student1

Name
Id Gender
Dept

Student2

Name
Id Gender
Dept

Student1 and **Student2** are variables of **StudentRecord** type.

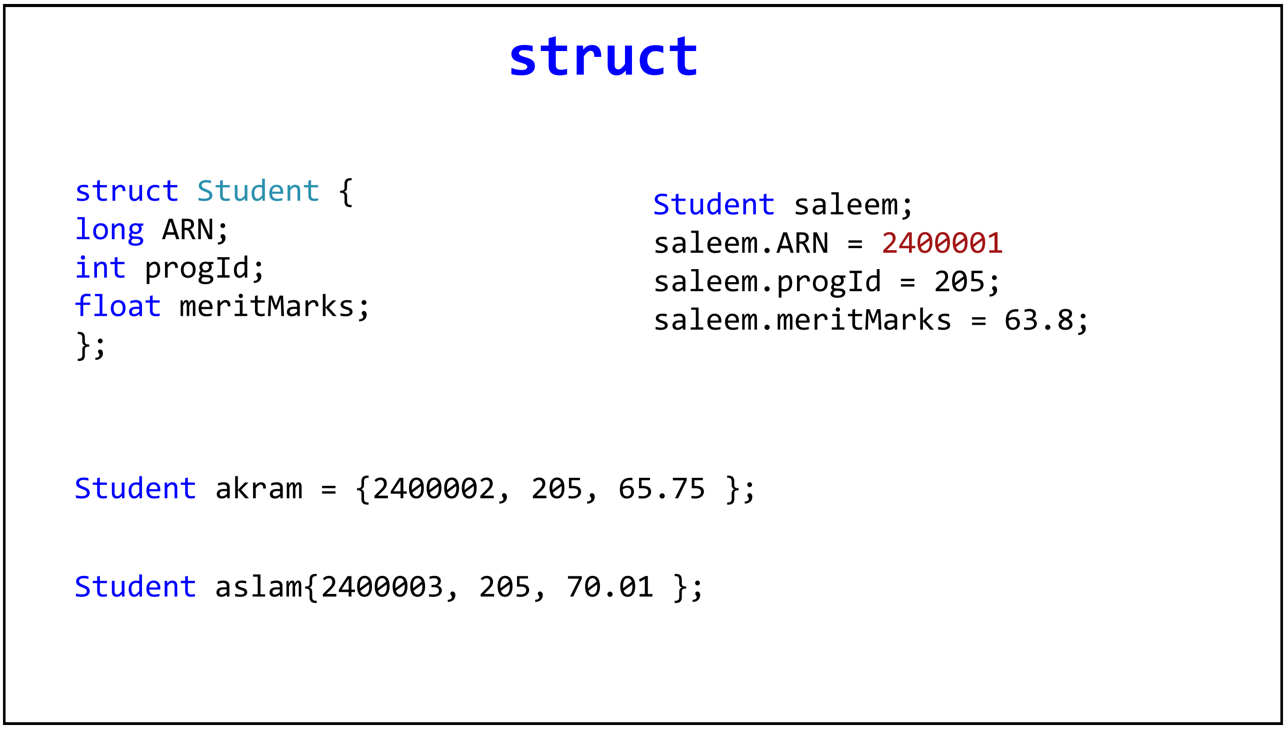


struct (Records)

- user-defined data type that groups related data elements of different types under a single name
- represent complex data structures, and organize related data

```
struct StructureName {
// DataType1 member1
// DataType2 member2
};
```

104



struct

```
struct Student {
long ARN;
int progId;
float meritMarks;
};
```

```
Student saleem;
saleem.ARN = 2400001
saleem.progId = 205;
saleem.meritMarks = 63.8;
```

```
Student akram = {2400002, 205, 65.75 };
```

```
Student aslam{2400003, 205, 70.01 };
```

105

Nested struct

```
struct Student {
long ARN;
float meritMarks;
};


struct Program {
int ID;
Student std;
};
```

```
Program MSDS;

MSDS.ID =  205;
MSDS.std.ARN = 2400001;
MSDS.std.meritMarks = 70.52;
```

Nested Structures

```
struct Distance
{
   int feet;
   float inches;
};

struct Room
{
   Distance length;
   Distance width;
};
```

```
void main(void)
{
   Room dinning;
   dinning.length.feet = 13;
   dinning.length.inches = 6.5;
   dinning.width.feet = 10;
   dinning.width.inches = 0.5
}
```

Structures and Pointers

```
struct Student {
long ARN;
int progId;
float meritMarks;
};


Student akram = {2400002, 205, 65.75 };

Student* ptrStudent = &akram;


cout << ptrStudent->ARN << endl;
```

108

Pointers to struct

```
struct Point
{
    int x;
    int y;
};
//----------------------------------------
void main()
{
    Point p1;
    Point* ptr;
    ptr = &p1;
    //Ways to access the elements of p1;

    p1.x = 10;
    p1.y = 5;

    ptr->x = 10; //Indirection operator
    ptr->y = 5;

    (*ptr).x = 10; //Deferencing ptr
    (*ptr).y = 5;

    _getch();
} // end main
```

109

struct

- C++ `struct` may have member functions like `class`
- constructors and destructors inside a `struct`
- By default, all members (including functions) of a `struct` are `public`, but you can define `private` or `protected` members using access specifiers

access modifier

Default access modifier:
In a **class**, members are **private** by default.
In a **struct**, members are **public** by default.

access modifier

```
struct MyStruct {
    int x;          // public by default

private:
    int y;          // explicitly private

public:
    void setY(int val) {
        y = val;      // accessible inside struct
    }
    int getY() {
        return y;
    }
};
```

```
main() {
    MyStruct s;
    s.x = 10;         // allowed (public)
    // s.y = 20;      // ❌ error (private)
    s.setY(20);       // allowed
    cout << s.x << " " << s.getY();
        }
```

Passing Structures to Function

```
struct part
{
     char partName[10];
     int partNumber;
     float cost;
};
void display(part);

//----------------------------------------
void main()
{
     part p1;
     cin >> p1.partName;
     cin >> p1.partNumber;
     cin >> p1.cost;
     display(p1);
     _getch();
} // end main

//^^^^^^^^^^^^^^^^^^^^^2^^^^^^^^^^^^^^^^^^^^^^^^^
//-------------------------------------
void display(part p2)
{
     cout << p2.partName;
     cout << p2.partNumber;
     cout << p2.cost;
}
```

Exercise:

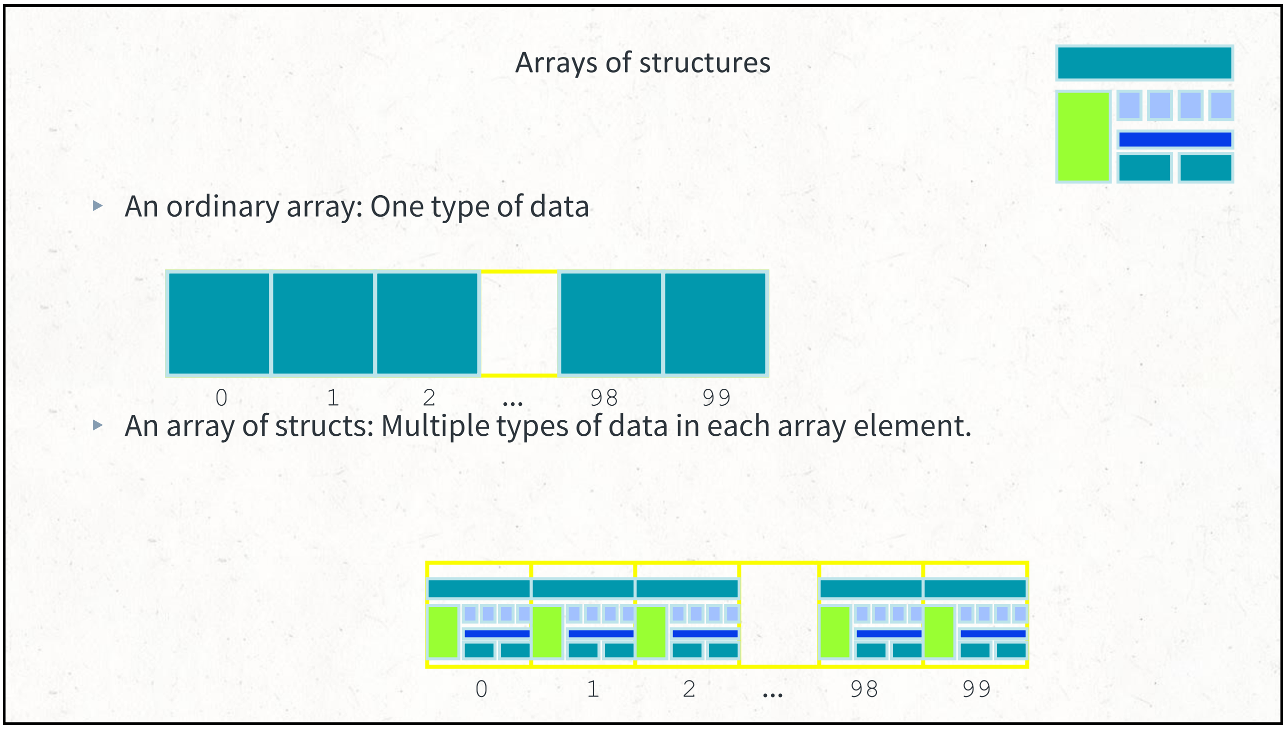
Array Rotation

1	2	3	4	5	6	7	8	9	10

8	9	10	1	2	3	4	5	6	7

Array of Structures

- Allows to create a collection of structure instances
- Useful for working with related data (like employee records, student details)

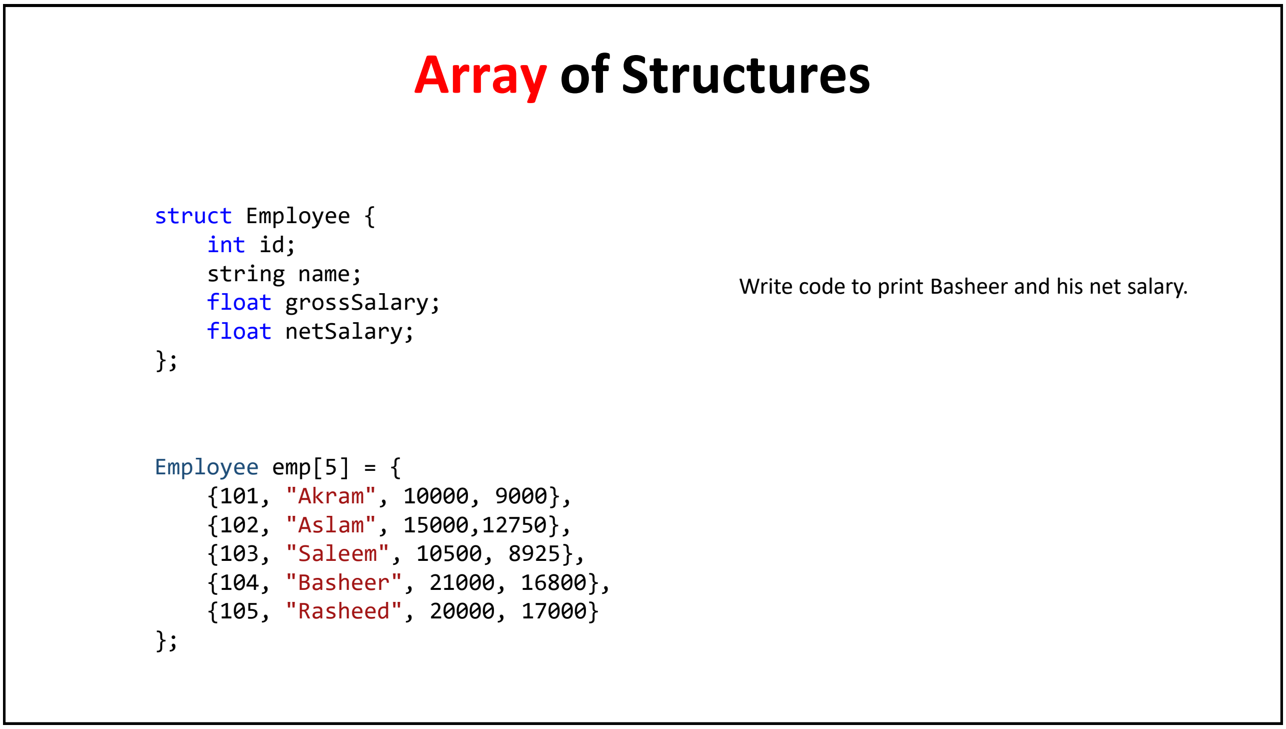


Arrays of structures

- An ordinary array: One type of data

- An array of structs: Multiple types of data in each array element.

116

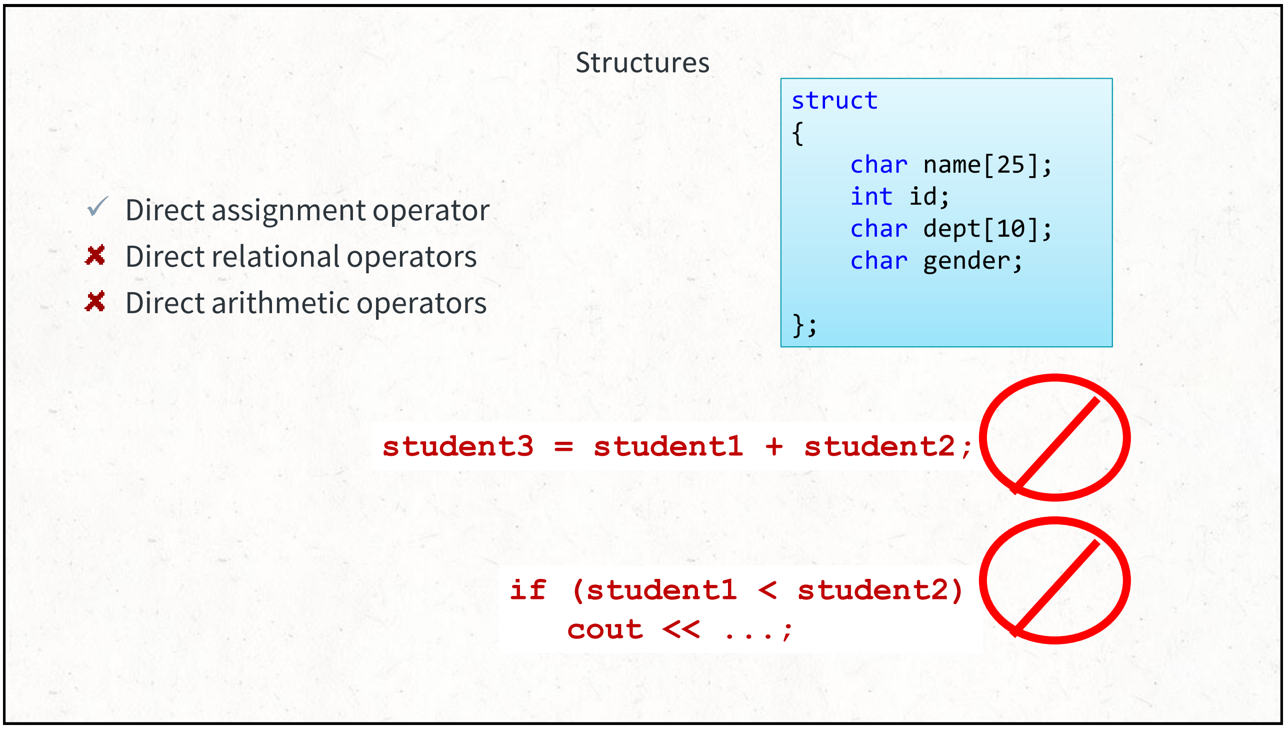


Array of Structures

```
struct Employee {
    int id;
    string name;
    float grossSalary;
    float netSalary;
};


Employee emp[5] = {
    {101, "Akram", 10000, 9000},
    {102, "Aslam", 15000,12750},
    {103, "Saleem", 10500, 8925},
    {104, "Basheer", 21000, 16800},
    {105, "Rasheed", 20000, 17000}
};
```

Write code to print Basheer and his net salary.

117



Structures

- ✓ Direct assignment operator
- ✗ Direct relational operators
- ✗ Direct arithmetic operators

```
struct
{
    char name[25];
    int id;
    char dept[10];
    char gender;

};
```

```
student3 = student1 + student2;
```

```
if (student1 < student2)
    cout << ...;
```