Module 6 Guidance Notes

Compound Data Types

(Arrays, Structures & Classes)

and File I/O

ENGG1340

Computer Programming II

Before We Start

- We will deal **mainly** with C++ in this module, and when C specific constructs are used, we will note the compiler settings.
- Important: We will be using the C++ 11 standard, so make sure that your compiler option is set appropriately. We suggest to use the following command to compile your C++ program:

```
g++ -pedantic-errors -std=c++11 your_program.cpp
```

The -pedantic-errors flag is to make sure that your code conforms to the ISO C/C++ standard. We will enforce this in your assignment submission too.
For more information about C/C++ standards, you may read https://en.wikipedia.org/wiki/ANSI_C and https://isocpp.org/std/the-standard

How to Use this Guidance Notes

- This guidance notes aim to lead you through the learning of the C/C++ materials. It also defines the scope of this course, i.e., what we expect you should know for the purpose of this course. (and which should not limit what you should know about C/C++ programming.)
- Pages marked with “Reference Only” means that they are not in the scope of assessment for this course.
- The corresponding textbook chapters that we expect you to read will also be given. The textbook may contain more details and information than we have here in this notes, and these extra textbook materials are considered references only.

How to Use this Guidance Notes

- We suggest you to copy the code segments in this notes to the coding environment and try run the program yourself.
- Also, try make change to the code, then observe the output and deduce the behavior of the code. This way of playing around with the code can help give you a better understanding of the programming language.

References

- cplusplus.com tutorial
  - Arrays
  - Character Sequences
  - Structures
  - File I/O
- Textbook Chapters
  - C++: How to program (9th edition)
    Electronic version available from HKU library
  - Ch. 14.1-5 (on file I/O)
  - Ch. 22.2-3 (on structs)

What are we going to learn?

Compound Data Types

- Arrays
- Structures
- Basic concept on Classes

File I/O

Part I

COMPOUND DATA TYPES

ARRAYS

Handling Data of the Same Type

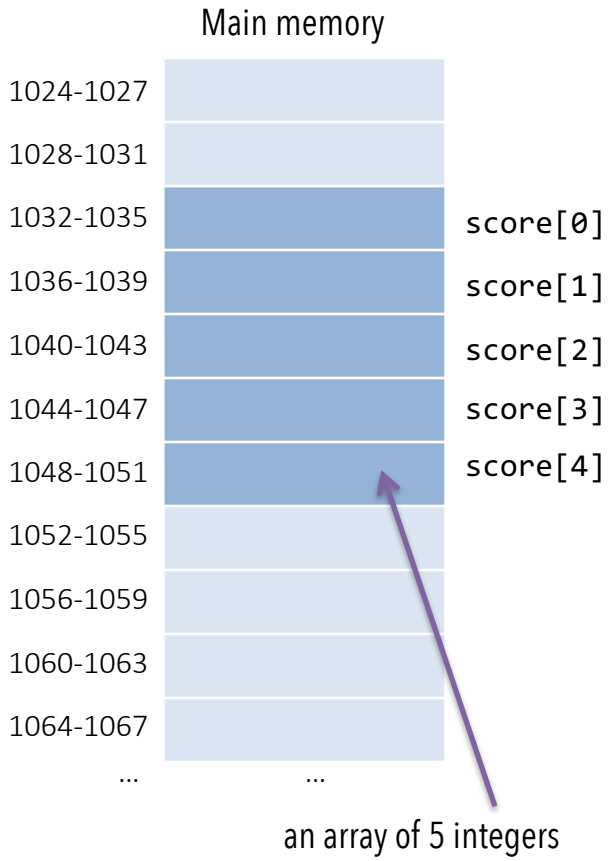
- Very often, a program needs to handle a collection of data of the same type
- Consider the following problem:
  - Write a program to input the scores of 80 students in a class and compute their average score and output those scores that are lower than the average.

```
int score_01, score_02, score_03, score_04, …, score_80;

cin >> score_01 >> score_02 >> … >> score_80;
double average = (score_01 + score_02 + … + score_80) / 80.0;

if (score_01 < average) cout << score_01 << endl;
if (score_02 < average) cout << score_02 << endl;
…
if (score_80 < average) cout << score_80 << endl;
```

Using individually named variables to handle such data is cumbersome, especially for large datasets

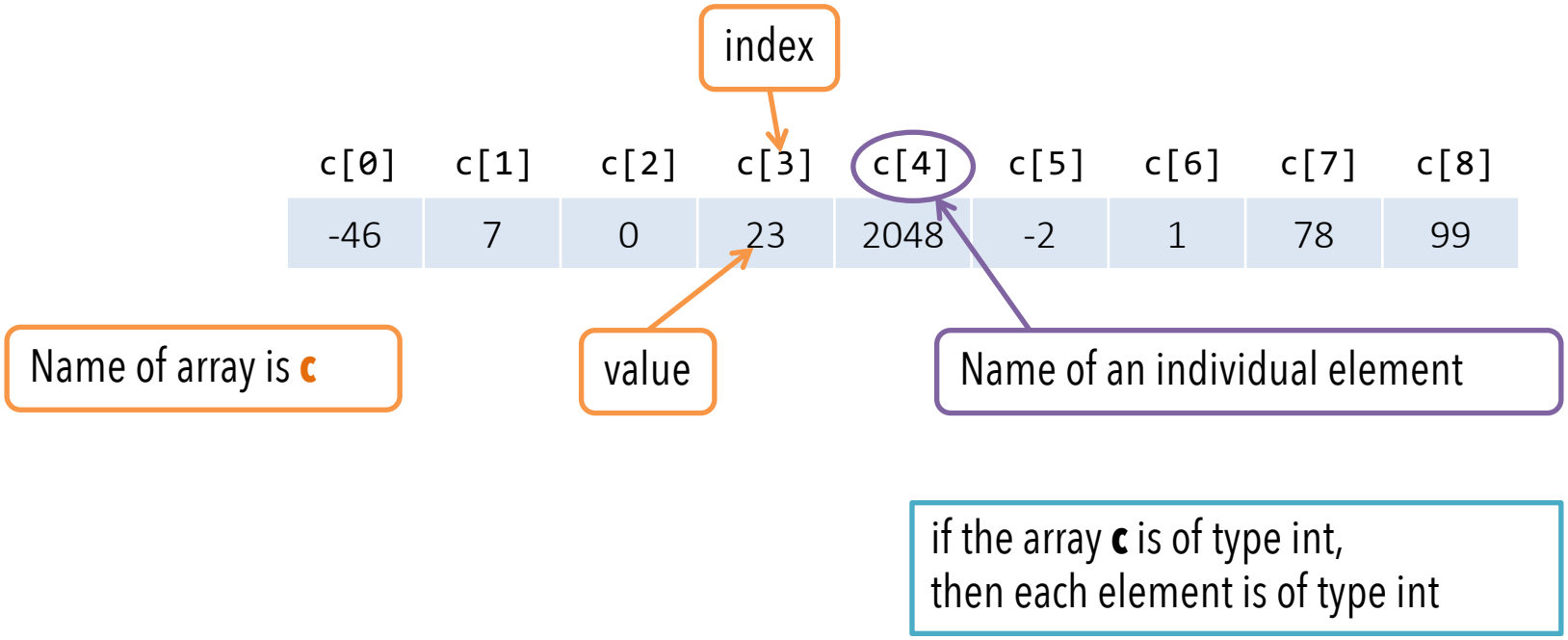
Arrays

- Arrays in C++ provide a convenient way to process such data
  - An array behaves like a list of variables (of the same type) with a uniform naming mechanism
  - An array is a consecutive group of memory locations that share the same type.



Arrays

- Each element of an array can be regarded as a variable of the base type, and can be accessed by specifying the name of the array and the position (index) in the subscript operator [ ]



Indexes of Array Elements

- Array indexes always start from zero and end with the integer that is one less than the size of the array

c[0]	c[1]	c[2]	c[3]	c[4]	c[5]
-46	7	0	23	2048	-2

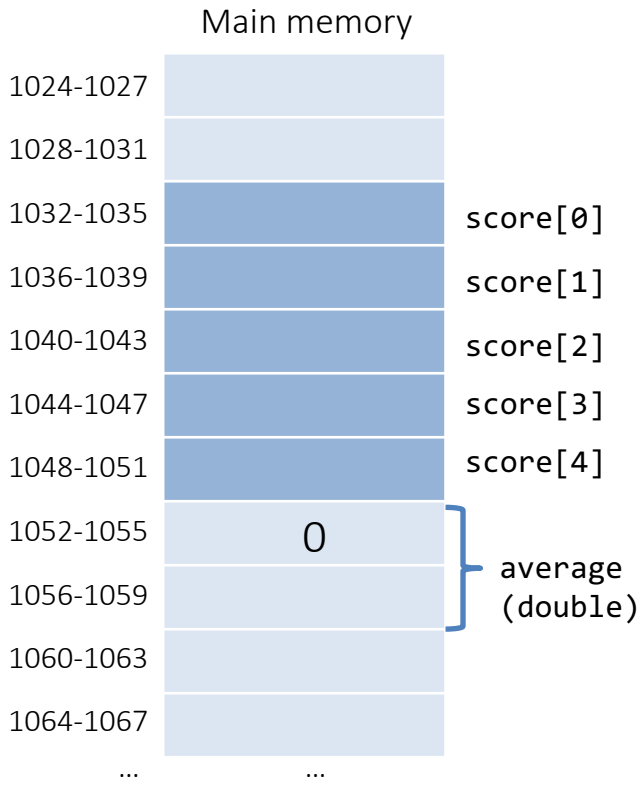
size of c is 6
elements are c[0], c[1], c[2], c[3], c[4], c[5]

- An array index can be any integer expression, including integer numerals and integer variables

```
c[1] = 100;
cout << c[0] + c[1] + c[2] << endl;
int x = c[6] / 2;

int a = 1, b = 2;
c[a + b] += 2;      // c[3] = c[3] + 2

int i = 4;
c[i + 1] = c[i] - 30;    // c[5] = c[4] - 30
```

Indexes of Array Elements

- The compiler will NOT report any error when an array index that is out of range is used
- On most systems, the program will proceed as if the index is legal and the memory cells corresponding to the nonexistent indexed variable will be accessed
- This may unintentionally change the values of the memory cells probably belonging to some other variables



size of `score` is 5
what if we write
`score[5] = 0`?
Try in a program and see what happens

Declaring an Array

- An array declaration specifies the base type, the name and the size of the array

Syntax

```
base_type    array_name[size];
```

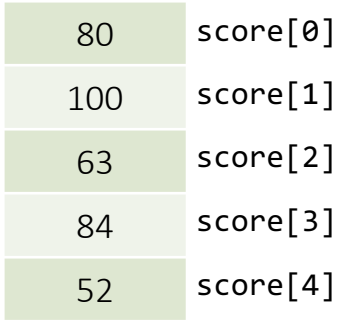
- Arrays are static entities in that their sizes cannot be changed throughout program execution
- Examples:

```
int score[5];           // an int array of 5 elements
char grade[8];          // a char array of 8 elements
double gpa[3];          // a double array of 3 elements

// Arrays and regular variables can be declared together
int max_score, min_score, score[5], passing_score;
```

Initialization with Initializer List

An array may be initialized in its declaration by using an equal sign followed by a list of values enclosed within a pair of braces { }

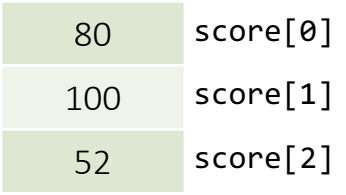
```
int score[5] = { 80, 100, 63, 84, 52 };
```



If an array is initialized in its declaration, the size of the array may be omitted and the array will automatically be declared to have the minimum size needed for the initialization values

```
int score[] = { 80, 100, 52 };
```

size equals 3



Initialization with Initializer List

- The compiler will report an error if too many values are given in the initialization, e.g.,

```
int score[5] = {80, 100, 63, 84, 52,96};
```

- It is, however, legal to provide fewer values than the number of elements in the initialization
  - Those values will be used to initialize the first few elements
  - The remaining elements will be initialized to a zero of the array base type

```
int score[5] = {80, 100};
```

80	score[0]
100	score[1]
0	score[2]
0	score[3]
0	score[4]

How to initialize all elements to have value 0?

Initialization with Initializer List

- It is illegal to initialize or change the content of the whole array using an equal sign after its declaration
- All the assignment statements below are therefore invalid

```
int score[5];

score = { 80, 100, 63, 84, 52 };
score[] = { 80, 100, 63, 84, 52 };
score[5] = { 80, 100, 63, 84, 52 };
```

Example: Print the contents of an array with a loop

Need this library for setw()

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

int main()
{
    // use initializer list to initialize array n
    int n[10] = { 32, 27, 64, 18, 95, 14, 90, 70, 60, 37 };

    cout << "Element" << setw(13) << "Value" << endl;

    // output each array element's value
    for ( int j = 0; j < 10; ++j )
        cout << setw(7) << j << setw(13) << n[j] << endl;

    return 0;
}
```

Using a loop to access and print out each element

setw(): set the width (i.e., # of space) for the next item to be printed out

Initialization with a Loop

- Use a loop to access each element and initialize them to some initial values.

using the loop control variable **i** as the index

Using a loop to access and print out each element

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

int main()
{
    int n[10];  // n is an array 10 integers

    // initialize elements of array n to 0
    for ( int i = 0; i < 10; ++i )
        n[i] = 0;   // set element at location i to 0

    cout << "Element" << setw(13) << "Value" << endl;

    // output each array element's value
    for ( int j = 0; j < 10; ++j )
        cout << setw(7) << j << setw(13) << n[j] << endl;

    return 0;
}
```

Using an Array

Compare the following two implementations.

- Write a program to input the scores of 80 students in a class and compute their average score and output those scores that are lower than the average.

```
int score_01, score_02, score_03, score_04, …, score_80;

cin >> score_01 >> score_02 >> … >> score_80;
double average = (score_01 + score_02 + … + score_80) / 80.0;

if (score_01 < average) cout << score_01 << endl;
if (score_02 < average) cout << score_02 << endl;
…
if (score_80 < average) cout << score_80 << endl;
```

New version using array

```
int total = 0, score[80], i;
for (i = 0; i < 80; ++i)
{
    cin >> score[i];
    total += score[i];
}
double average = total / 80.0;
for (i = 0; i < 80; ++i)
    if (score[i] < average) cout << score[i] << endl;
```

Example 1

- To specify an array's size with a constant variable and to set array elements with calculations

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

int main()
{
    // constant variable can be used to specify array size
    const int arraySize = 10;

    int s[arraySize];   // array s has 10 elements

    for (int i = 0; i < arraySize; ++i)     // set the values
        s[i] = 2 + 2*i;

    cout << "Element" << setw(13) << "Value" << endl;

    // output contents of array s in tabular format
    for ( int j = 0; j < arraySize; ++j )
        cout << setw(7) << j << setw(13) << s[j] << endl;

    return 0;
}
```

Only need to change the value of `arraySize` to make the program scalable, i.e., for the program to work for other array sizes.

Example 2

- Using array elements as counters, e.g., roll a die and record the frequency of occurrences for each side.
- If **`frequency[i]`** stores the number of occurrences of face **`i`**, then what is the array size needed for storing the frequencies?

```
int  frequency[ 7 ];
// ignore element 0, use elements 1, 2, …, 6 only
```

- How to simulate a die-rolling?

Use a random number generator to generate a random number within [1..6] using the expression **`rand() % 6 + 1`**

Example 2

```
#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <ctime>
using namespace std;

int main()
{
    const int arraySize = 7;            // ignore element zero
    int frequency[arraySize] = {};      // initialize elements to 0

    srand( time(0) );   // seed random number generator

    // roll die 6,000,000 times; use die value as frequency index
    for (int roll = 1; roll <= 6000000; ++roll)
        ++frequency[ 1 + rand() % 6 ];

    cout << "Face" << setw(13) << "Frequency" << endl;

    // output each array element's value
    for ( int face = 1; face < arraySize; ++face )
        cout << setw(4) << face << setw(13) << frequency[face]
                        << endl;

    return 0;
}
```

Exercises

1. Write a program to initialize an array with the integers 1-10 and compute the sum of the 10 numbers.
2. Write a program to initialize an array with the first 10 odd integers starting from 1, and compute the product of the 10 numbers.
3. Write a program to initialize an array with the 10 characters ‘a’ to ‘j’ and print them out in reverse.
4. Write a program to get 10 input numbers from the users, print them out in reverse, and print out their sum.
5. Write a program to get input integers from the user repeatedly until the user enters 0. Your program should count the number of 1, 2, 3, 4, 5, 6 input by the user and print the frequencies out.

* Compare question 5 to the dice-rolling example in the previous slide.

Passing Array Elements to Functions

- Like regular variables, array elements can be passed to a function either by value or by reference.

```
// returns the square of an integer
int square( int x )
{
    return x * x;
}
```

Pass by value

To square each entry of an array

```
int a[4] = { 0, 1, 2, 3 };

for (int i = 0; i < 4; ++i)
{
    a[i] = square( a[i] );

}
```

Passing Array Elements to Functions

- Like regular variables, array elements can be passed to a function either by value or by reference.

```
// returns the square of an integer
void square( int &x )
{
    x *= x;
}
```

Pass by reference

To square each entry of an array

```
int a[4] = { 0, 1, 2, 3 };

for (int i = 0; i < 4; ++i)
{
    square( a[i] );
}
```

Passing Arrays to Functions

- It is also possible to pass an entire array to a function (called an array parameter)
- To indicate that a formal parameter is an array parameter, a pair of square brackets [ ] is placed after its identifier in the function header and function declaration

Syntax (function header)

```
type_ret    func_name(base_type array_para[], ...)
```

Syntax (function declaration)

```
type_ret    func_name(base_type array_para[], ...);
```

Passing Arrays to Functions

- Examples

Function definition

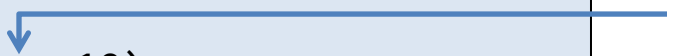
```
void modifyArray( int b[], int arraySize )
{
    …
}
```

Function declaration (function prototype)

```
void modifyArray( int [], int);
```

Function call

```
int a[10];
modifyArray( a, 10);
```



Just need the array name here; no square brackets after the array identifier in function call

Passing Arrays to Functions

- An array parameter behaves very much like a pass-by-reference parameter
  - The call functions can modify the element values in the callers' original arrays.
- An array argument only consists of the array identifier, but does not provide information of its size
  - C++ **does not perform check** on the array bound, so we may pass an array of any size to a function
  - Another int argument is often used to tell the function the size of the array

Passing Arrays to Functions

```
int main()
{
    const int arraySize = 5; // size of array a
    int a[ arraySize ] = { 0, 1, 2, 3, 4 }; // initialize array a

    cout << "Effects of passing entire array:"
         << "\nThe values of the original array are:\n";

    // output original array elements
    for ( int i = 0; i < arraySize; ++i )
        cout << setw( 3 ) << a[ i ];

    cout << endl;

    // pass array a to modifyArray
    modifyArray( a, arraySize );
    cout << "The values of the modified array are:\n";

   // output modified array elements
    for ( int j = 0; j < arraySize; ++j )
        cout << setw( 3 ) << a[ j ];

    return 0;
}
```

See definition of modifyArray on the next slide

Passing Arrays to Functions

```
// in function modifyArray, "b" points to the
// original array "a" in memory
void modifyArray( int b[], int sizeOfArray )
{
    // multiply each array element by 2
    for ( int k = 0; k < sizeOfArray; ++k )
        b[ k ] *= 2;
}
```

```
Effects of passing entire array:
The values of the original array are:
  0  1  2  3  4
The values of the modified array are:
  0  2  4  6  8
```

Screen output

* Note that the values of the array elements **are modified** by the function, which is of a similar effect as pass-by-reference

Searching an Array

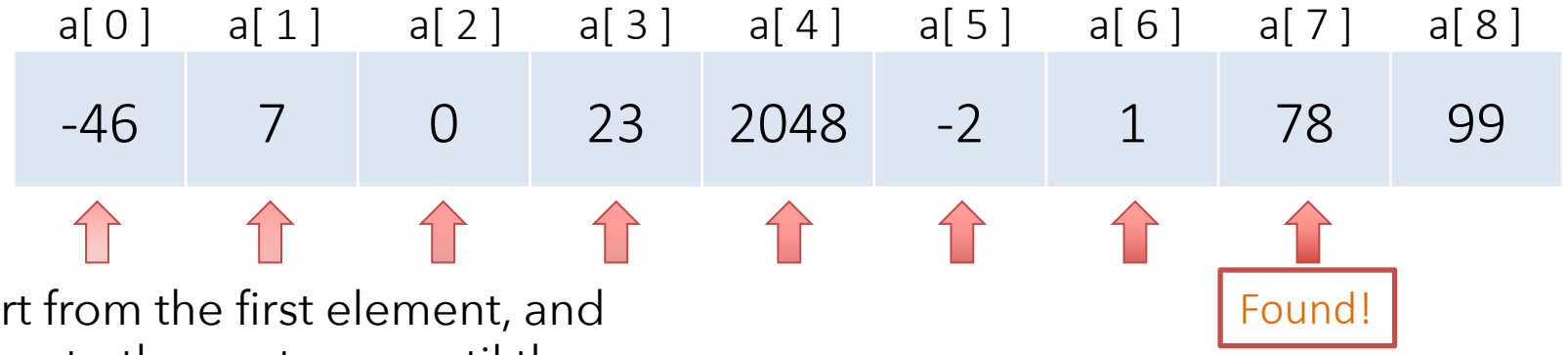
- A common programming task is to search an array for a given value

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]
-46	7	0	23	2048	-2	1	78	99

- Where is the item "78"? At index 7
- Where is the item "100"? Not found
- If the value is found, the index of the array element containing the value is returned
- If the value is not found, –1 is returned

Linear Search

- The simplest method is to perform a linear search in which the array elements are examined sequentially from first to last



Start from the first element, and move to the next one, until the target item (78) is found.

How many elements need to be examined on average? Half of the array

How many elements need to be examined for the worst case? Entire array

Linear Search

```
// linear search of key value in array[]
// return the index of first occurrence of key in array[]
// return -1 if key is not found in array[]
int linearSearch( const int array[], int sizeOfArray, int key )
{
    for ( int j = 0; j < sizeOfArray; ++j )
        if ( array[ j ] == key ) // if found,
            return j;            // return location of key

    return -1;  // key not found
}
```

`const int array[]`: the `const` keyword is to specify that the contents of the formal parameter `array[]` are to remain constant (i.e., not to be changed) in this function.

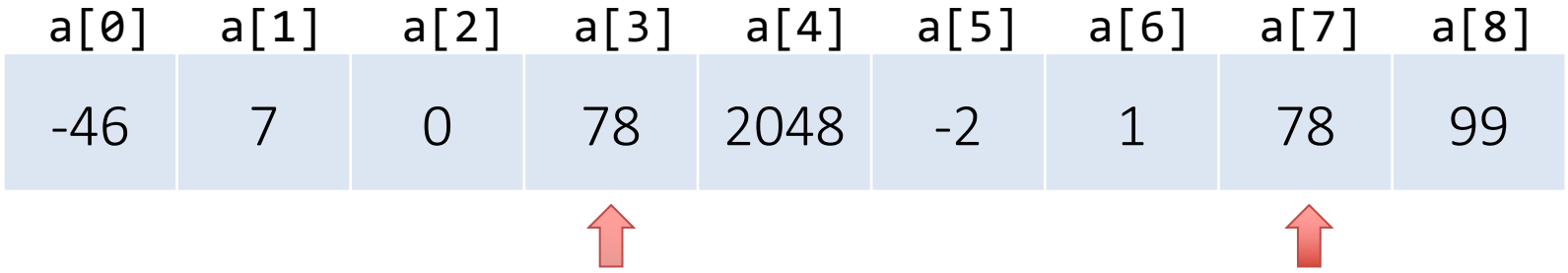
Linear Search

search.cpp

```
int main()
{
    const int arraySize = 10;  // size of array
    int a[ arraySize ];         // declare array a
    int searchKey;              // value to locate in array a

    // fill in some data to array
    for ( int i = 0; i < arraySize; ++i )
        a[i] = 2 * i;

    cout << "Enter an integer to search: ";
    cin >> searchKey;

    // try to locate searchKey in a
    int element = linearSearch( a, arraySize, searchKey );

    // display search results
    if ( element != -1 )
        cout << "Value found in element " << element << endl;
    else
        cout << "Value not found" << endl;

    return 0;
}
```

Linear Search (Variant)

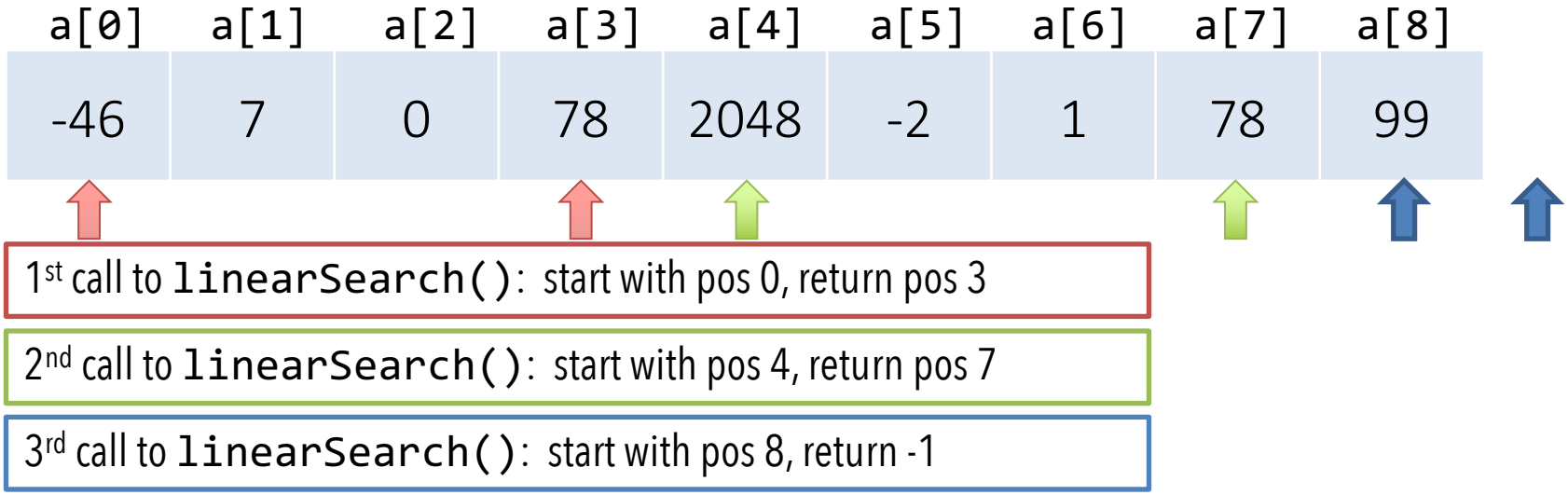
- The function **`linearSearch()`** returns only the first occurrence of the search item.
- What if we need the locations of ALL occurrences of the search item?



If search item = 78,
the program should be able to identify positions 3 and 7.

Linear Search (Variant)

- How to make changes to **`linearSearch()`** so that we can make use of it to look for all occurrences of an item?
- What does **`linearSearch()`** return?
- How about if we start searching from the returned position of a previous call of **`linearSearch()`**?



Linear Search (Variant)

- Function prototype for new **`linearSearch()`**

```
// linear search of key value in array[]
// starting search from startPos
// return the index of first occurrence of key in array[]
// return -1 if key is not found in array[]
int linearSearch( const int array[], int sizeOfArray,
                  int key, int startPos  );
```

- The **`main()`** function also needs some modification, so that **`linearSearch()`** will be called repeatedly until no more search item can be found.

Sorting an Array

- Another most widely encountered programming task is to sort the values in an array, e.g., in ascending/descending order
- There are many different sorting algorithms, e.g., insertion sort, bubble sort, quicksort, etc.
- One of the easiest sorting algorithms is called selection sort

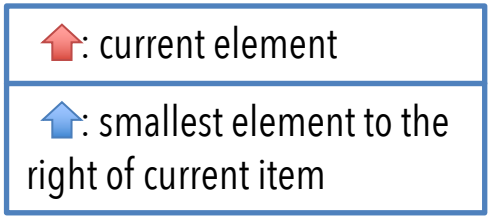
	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]
before	-2	7	0	23	2048	-46
after sorting in ascending order	-46	-2	0	7	23	2048

Selection Sort

- A total of N iterations are needed to sort N elements
- At each iteration **i**, i = 0, …, N-1,
  - exchange **a[i]** with the **smallest** item among a[i]… a[N-1] (or the largest, if sort in descending order)

- An important property is that, after each iteration **i**
  - the elements from a[0]…a[i] are sorted,
  - the elements from a[i+1]..a[N-1] remain to be sorted

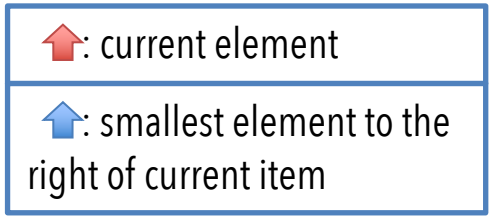
Selection Sort



To sort in ascending order

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	
	-2	7	0	23	2048	-46	
Iteration 0 (look for smallest element from a[0] to a[5], and swap with a[0])	-2	7	0	23	2048	-46	before
	-46	7	0	23	2048	-2	after
Iteration 1 (look for smallest element from a[1] to a[5], and swap with a[1])	-46	7	0	23	2048	-2	before
	-46	-2	0	23	2048	7	after
Iteration 2 (look for smallest element from a[2] to a[5], and swap with a[2])	-46	-2	0	23	2048	7	before
	-46	-2	0	23	2048	7	after

Selection Sort



To sort in ascending order

	a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	
Iteration 3 (look for smallest element from a[3] to a[5], and swap with a[3])	-46	-2	0	23	2048	7	before
	-46	-2	0	7	2048	23	after
Iteration 4 (look for smallest element from a[4] to a[5], and swap with a[5])	-46	-2	0	7	2048	23	before
	-46	-2	0	7	23	2048	after
Iteration 5 (look for smallest element from a[5] to a[5], and swap with a[5])	-46	-2	0	7	23	2048	before
	-46	-2	0	7	23	2048	after

Selection Sort

```
// sort values in array[] in ascending order by selection sort
void sort(int array[], int sizeOfArray )
{
     int i, j, idx;
     int min;

     for ( i = 0; i < sizeOfArray; ++i )
     {
          min = array[i];
          idx = i;
          for ( j = i + 1; j < sizeOfArray; ++j )
          {
               if ( array[j] < min )
               {
                    min = array[j];
                    idx = j;
               }
          }
          if ( idx != i )
               swap( array[i], array[idx] );    // swap values
     }
}
```

```
void swap(int &a, int &b)
{
     int tmp = a;
     a = b;
     b = tmp;
     return;
}
```

Find the minimum from array[i] to array[N-1]

⬆ : array[i]

⬆ : array[idx]

Selection Sort

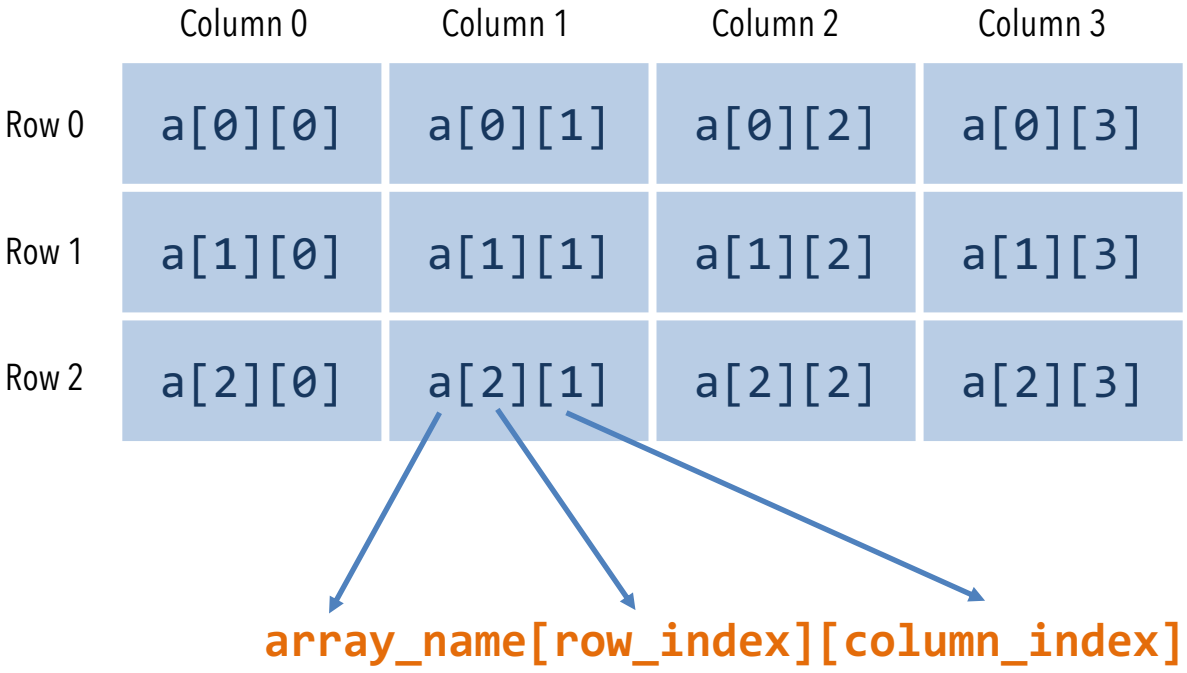
sort.cpp

```
int main()
{
    const int arraySize = 6;                          // size of array
    int a[ arraySize ] = {-2, 7, 0, 23, 2048, -46};   // declare array a

    cout << "Original array: ";
    print_array( a, arraySize );

    sort( a, arraySize );

    cout << "Sorted array: ";
    print_array( a, arraySize );

    return 0;
}
```

```
void print_array( const int array[], int sizeOfArray )
{
    for ( int i = 0; i < sizeOfArray; ++i )
        cout << "[" << setw(2) << i << "] ";
    cout << endl;

    for ( int i = 0; i < sizeOfArray; ++i )
        cout << setw(3) << array[i] << "  ";
    cout << endl;
}
```

Two-Dimensional Arrays

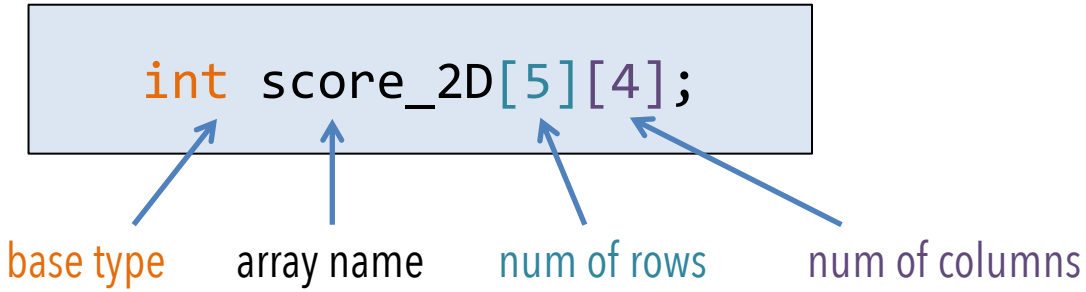
- How about a table of values arranged in rows and columns?
- A two-dimensional array (2D array):



A 2D array with
3 rows and
4 columns
(a **3-by-4 array**)

Two-Dimensional Arrays

- To declare a 2D array:



- Similar to the 1D case, each indexed variable of a multi-dimensional array is a variable of the base type, e.g.,

```
int score_2D[5][4];
score_2D[0][0] = 80;
score_2D[4][3] = score_2D[0][0] + 20;
```

Two-Dimensional Arrays

- Initialization:

```
int b[2][3] = { 1, 2, 3, 4, 5 };
```

fill up values for 1st row first, then 2nd row

b	0	1	2
0	1	2	3
1	4	5	0

Two-Dimensional Arrays

- Using a nested for loop to run through all elements.

```
const int nRows = 3;
const int nCols = 5;

int array2D[nRows][nCols];
int i, j;

// assign initial values
for (i = 0; i < nRows; ++i)
   for (j = 0; j < nCols; ++j)
         array2D[i][j] = nCols*i + j;
```

array2D.cpp

```
// print out array contents
for (i = 0; i < nRows; ++i)
{
   for (j = 0; j < nCols; ++j)
      cout << setw(3) << array2D[i][j] << ' ';
   cout << endl;       // start new line for each row
}
```

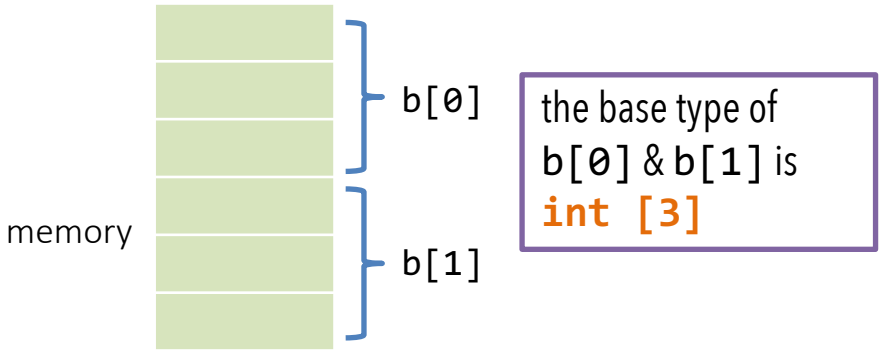
Reference Only

Multi-Dimensional Arrays

- Arrays with two or more dimensions are known as multi-dimensional arrays.

e.g.
```
int score_3D [5][4][3];
```

- A multi-dimensional array is an array of arrays.
  - All array elements are stored consecutively in memory, regardless of the number of dimensions.
  - E.g., **`int b[2][3]`** is a 1D array of size 2, with each element being a 1D integer array of size 3.



2D Array as Function Parameter

- Recall that for using a 1D array as parameter:

```
void print_1D_array ( int array [], int sizeOfArray );
```

indicate that this is an array of **int**

- When a 2D array parameter is used in a function header or function declaration, the size of the first dimension is not given, but the remaining dimension size must be given in square brackets.
- Now for using a 2D array as parameter:

```
void print_2D_array ( int array [][5], int numRows);
```

indicate that this is an array of **int[5]**

2D Array as Function Parameter

```
int main()
{
    const int nRows = 3;
    const int nCols = 5;

    int array2D[nRows][nCols];
    int i, j;

    // assign initial values
    for (i = 0; i < nRows; ++i)
        for (j = 0; j < nCols; ++j)
            array2D[i][j] = nCols*i + j;

    print_2d_array( array2D, nRows );

    return 0;
}
```

```
void print_2d_array( const int a[][5], int numRows)
{
    // print out array contents
    for (int i = 0; i < numRows; ++i) {
        for (int j = 0; j < 5; ++j)
            cout << setw(3) << a[i][j] << ' ';
        cout << endl;          // start new line for each row
    }
}
```

array2D_func.cpp

CHAR & CHAR ARRAY

char Data Type

- Recall that the data type char is used for representing single characters, e.g., letters, digits, special symbols.

```
char c1 = 'a';     // the character 'a'
char c2 = '2';     // the character '2'
char c3 = '\n';    // the newline character
```

- Each char takes up **1 byte** of storage space.
- The most commonly used character set is ASCII (American Standard Code for Information Interchange), which uses 0-127 to represent a character.

The ASCII Character Set

Control Characters	
0	(Null character)
1	(Start of Header)
2	(Start of Text)
3	(End of Text)
4	(End of Trans.)
5	(Enquiry)
6	(Acknowledgement)
7	(Bell)
8	(Backspace)
9	(Horizontal Tab)
10	(Line feed)
11	(Vertical Tab)
12	(Form feed)
13	(Carriage return)
14	(Shift Out)
15	(Shift In)
16	(Data link escape)
17	(Device control 1)
18	(Device control 2)
19	(Device control 3)
20	(Device control 4)
21	(Negative acknowl.)
22	(Synchronous idle)
23	(End of trans. block)
24	(Cancel)
25	(End of medium)
26	(Substitute)
27	(Escape)
28	(File separator)
29	(Group separator)
30	(Record separator)
31	(Unit separator)
127	(Delete)

Printable Characters					
32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

char and **int**

- Examples

```
char c = 'A';
cout << c << endl;
```

Screen output

```
A
```

```
char c = 65;
cout << c << endl;
```

Screen output

```
A
```

Since the data type of c is char, assigning an integer to c is treated as assigning an ASCII code to c

char and **int**

- We may use an int variable to store the value of a char variable. In this case, the ASCII code of the char will be stored.

```
char letter = 'A';
int val = letter;

cout << letter << endl;
cout << val << endl;
```

Screen output

```
A
65
```

char and **int**

- Arithmetic operations between char variables indeed operates on the ASCII values of the characters.

```
char letter1 = 'a';
char letter2 = 'b';
cout << letter1 << endl;
cout << letter2 << endl;

cout << letter1 - letter2 << endl;
cout << 'z' - 'a' << endl;

letter2--;
cout << letter2 << endl;
```

Screen output

```
a
b
-1
25
a
```

char and **int**

- More examples

```
char c = '1';
int num = c + 1;
cout << num << endl;
```

Screen output

```
50
```

The statement `int num = c + 1` takes the ASCII value of `'1'` (i.e., 49) for the addition operation.

```
char from = 'd';
char to = from – ('a' – 'A');
cout << to << endl;
```

Screen output

```
D
```

This is a technique to convert a small letter to its corresponding capital letter. The expression `'a' – 'A'` tells the difference in ASCII values between a small letter and its capital letter.

Comparisons for **char** Data Type

- How to determine if a letter is in lowercase or uppercase?

```
char letter;
cin >> letter;

if (    letter >= 'a' && letter <= 'z'    )
   cout << letter << " is in lowercase." << endl;
else if (   letter >= 'A' && letter <= 'Z'    )
   cout << letter << " is in uppercase." << endl;
```

Since the ASCII codes of the small letters and the capital letters are in order, we may use the relational operators (<, >, <=, >=) and equality operators (==, !=) to compare between characters.

Text as Strings

- How about if we want to represent a sequence of characters?

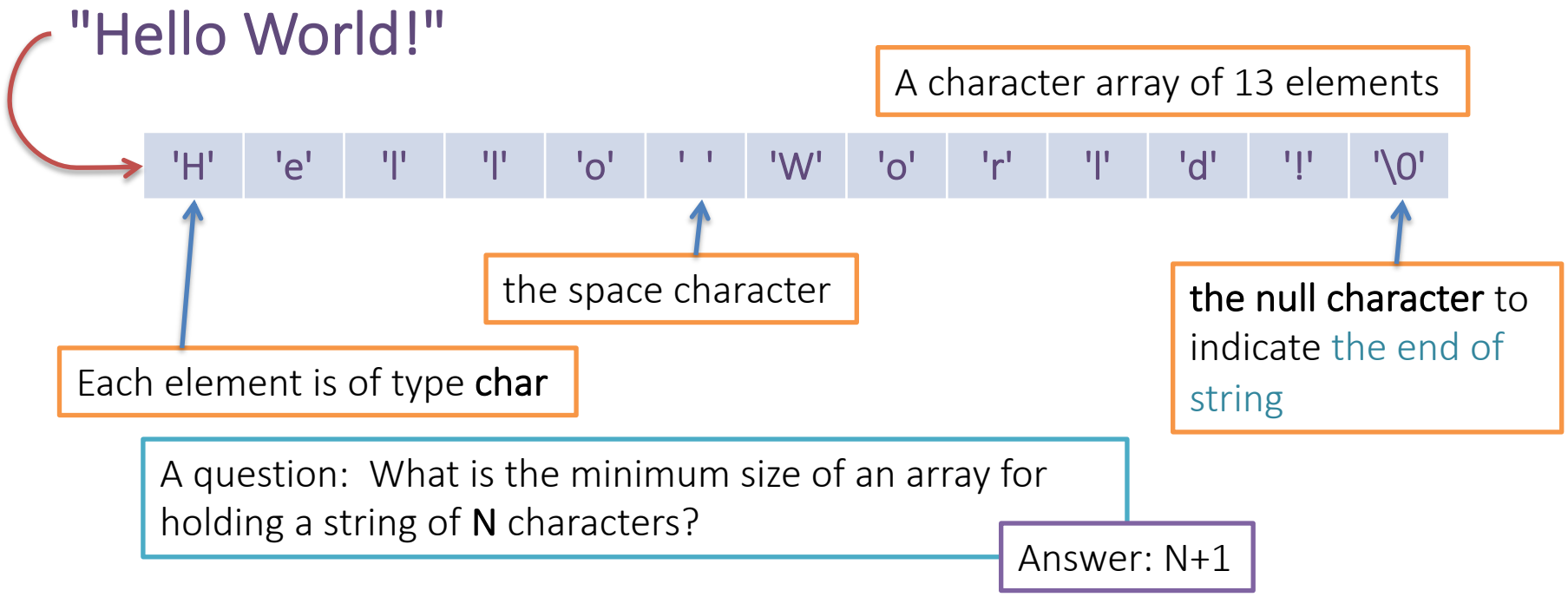
```
cout << "Hello World!" << endl;
```

- Strings are a sequence of characters and in C++ we use a pair of double quotation marks to enclose a string.

```
"Hello World!"
"ENGG1112"
"@_@"
```

C-Strings (Character Arrays)

- A common representation of a string is the **C-Strings**.
- A C-string is stored as an **array of char** (i.e., a character array), and is ended by a **null character** ('\0').



C-Strings (Character Arrays)

- What is the difference between `'A'` and `"A"`?

a char `'A'`

a string `"A"` containing
two chars `'A'` and `'\0'`

- Declaring a character array and assign a string to it:

```
char name[16] = { 'J', 'o', 'h', 'n', '\0'};
```

- Examples:

```
char name[16] = { 'J', 'o', 'h', 'n', '\0'};
cout << name;
```

```
John
```

Screen output

C-Strings (Character Arrays)

- We may also have the following declarations for C-strings:

```
char name[16] = "John";
```

1

```
char name[] = "John";
```

2

What's the difference between the above two declarations?

In 1, the size of the array name is of 16 chars; and in 2, the size is of 5 chars.

C-Strings (Character Arrays)

- Like regular arrays, it is not possible to copy blocks of data to a character array using an equal sign (i.e., an assignment) after its declaration.
- Hence, all the assignment statements below are invalid.

```
char name[16];

name = { 'J', 'o', 'h', 'n', '\0' };
name[] = { 'J', 'o', 'h', 'n', '\0' };

name = "John";
name[] = "John";
```

The Null character

- What is the output of the following program segment?

```
char name[] = "Steve";
cout << name << endl;

name[5] = 'n';
cout << name << endl;
```

Screen output

```
Steve
Steven??@#v
```

The null character at `name[5]` is overwritten and hence we have an unexpected end of string.

We may access each individual character using the subscript operator [], just as for an ordinary array.

Working with C-Strings

- cout and cin can be used for I/O for C-strings:

```
char msg[] = "Please enter your name: ";
char name[80];

cout << msg;
cin >> name << endl;

cout << "Hello " << name << "!" << endl;
```

```
Please enter your name:
Steve
Hello Steve!
```

Screen output

- Side-notes only: C++ provides a set of functions for C-string manipulation, e.g., string copy **strcpy()**, string compare **strcmp()**, string length **strlen()**, under the **cstring** header

Exercises

1. Write a function `charToInt` that will take a `char` integer and returns an `int`.
   - E.g., `charToInt('9')` will return `9`

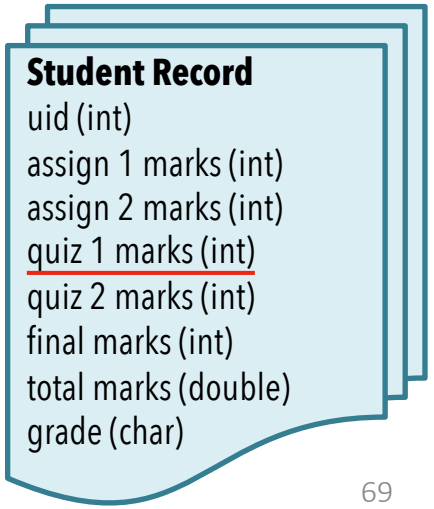
2. Write a function `toUpper` that will take a lower case char and returns its upper case.
   - E.g., `toUpper('a')` will return `'A'`

3. Write a function `toUpper2` that will take a lower case char array and change it to its upper case equivalent.
   - You may assume that the char array is filled with chars from `'a'` to `'z'`

STRUCTURES

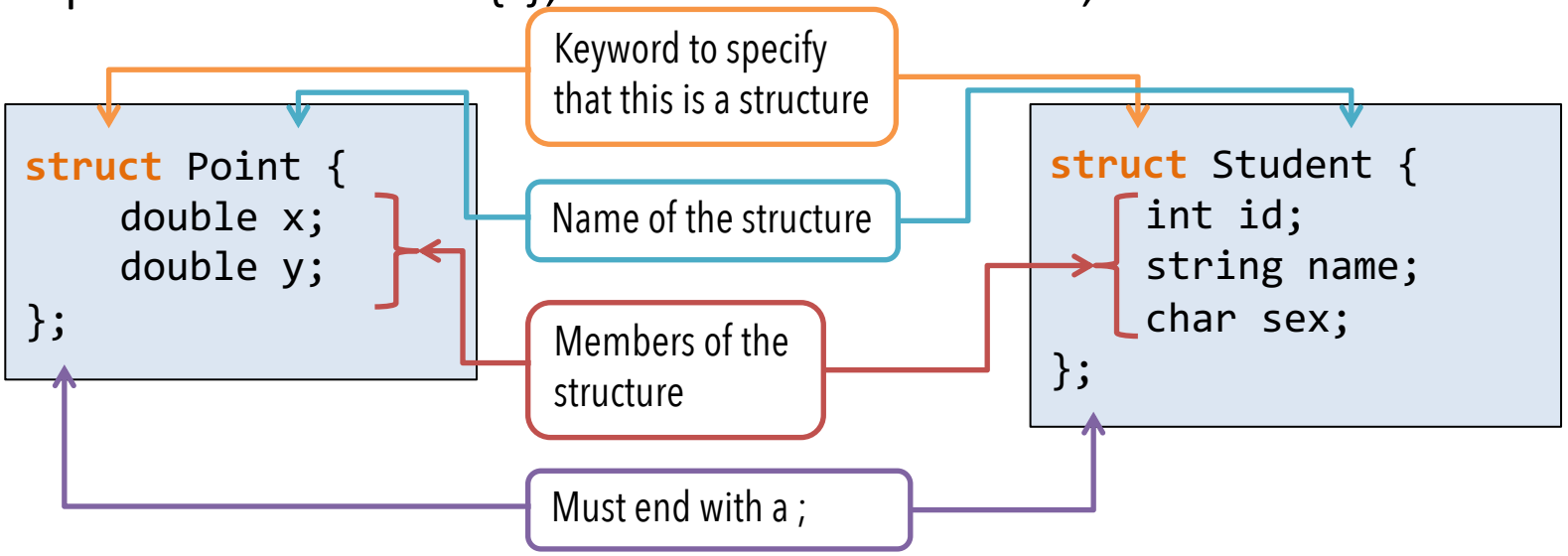
Structures

- A structure is a collection of one or more variables grouped together under a single name
- The data elements in a structure are known as its member variables (or simply members), which can be of different types
- Structures help organizing complex data
  - Allow a group of related variables to be treated as a single unit instead of separate entities
- Structures act like any basic data type
  - May be copied and assigned to variables
  - May be passed to and returned by functions



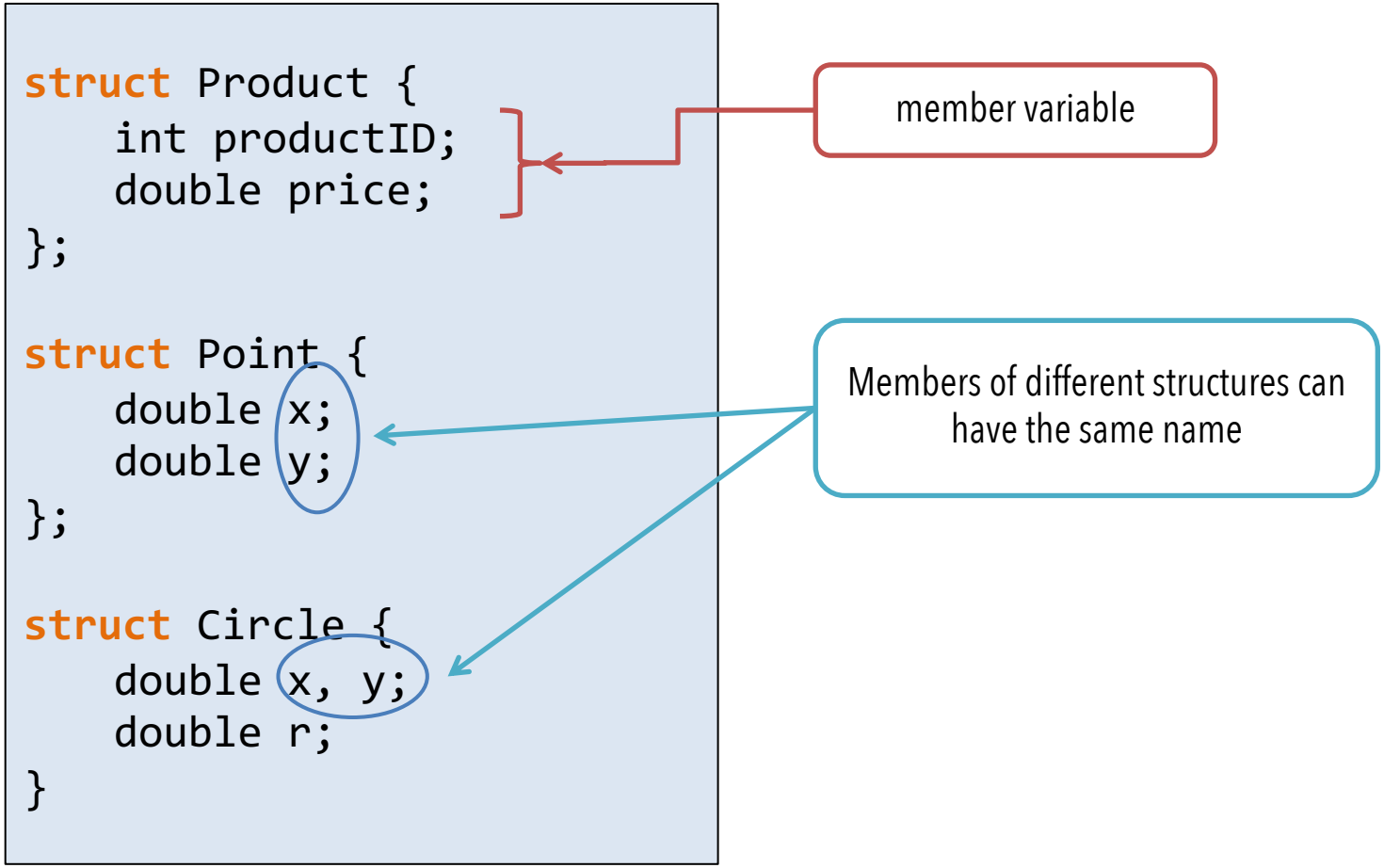
Definition

- In C++, a structure is defined using the keyword **struct**, followed by a structure tag, a list of member variables (with types and identifiers) enclosed within a pair of braces { }, and a semicolon ;



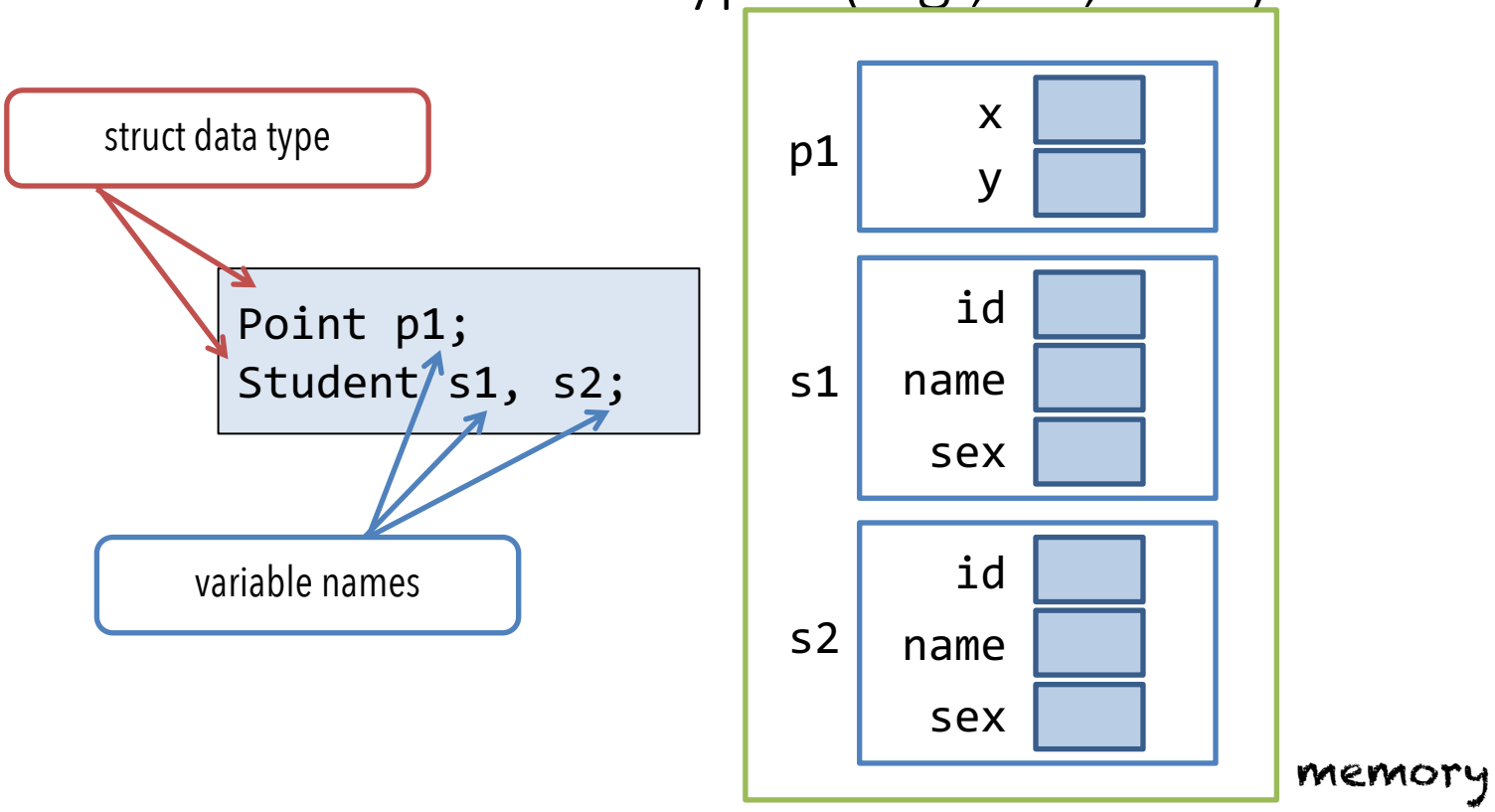
Definition

- Examples



Declaration

- Structure variables can be declared just as what you do for the basic data types (e.g., int, char)



Initialization

- A structure variable can be initialized in its declaration:

```
Point p1 = { 1.0, 2.0 };
Student s1 = { 301323549, "Amy Siu", 'F' };
Student s2 = s1;
```

Can be initialized with another variable of the same structure data type

Order of the members must be the same as that specified in the definition

```
Point p2 = { 1.0, 2.0, 3.0 };
```

A compilation error will be generated, since there are more values than the number of members

```
Point p3 = { 1.0 };
```

There are fewer values than the number of members, remaining variables are set to zero of their data type.
`(x = 1.0, y = 0.0 )`

Member Variables

- A member variable can be used just as other variables of the basic data types
- We may use the dot operator . to access the member variables of a structure

```
Point pt1 = { 1.0, 2.0 };
Point pt2 = pt1;

pt1.x *= 2.0;    // pt1.x = pt1.x * 2.0
pt1.y /= 2.0;    // pt1.y = pt1.y / 2.0
pt2.x++;             // pt2.x = pt2.x + 1
pt2.y--;             // pt2.y = pt2.y - 1
```

What are the values of all the member variables?

result

```
pt1.x = 2.0
pt1.y = 1.0

pt2.x = 2.0
pt2.y = 1.0
```

the dot operator

```
Student s1 = { 301323549, "Amy Siu", 'F' };
int l = s1.name.length();
```

What is the value of l?

```
l = 7
```

a string variable

Member Variables

- Example

```
struct Student {
    int id;
    string name;
    char sex;
    double GPA;
};

Student s1;
```

What is the data type of each of the following?

- `s1.id` — `int`
- `s1.sex` — `char`
- `s1.name` — `string`
- `s1` — `Student`
- `Student.GPA` — invalid. `Student` is a data type, not a variable
- `s2.GPA` — invalid. `s2` is undeclared.

Operators

- Structure variables do not work with arithmetic (+/−), relational (>/<), equality (==) and logical operators (&&/||) by default
  - because struct is user-defined
- All expressions below are therefore invalid

```
Point pt1 = {1.0, 2.0}, pt2 = {3.0, 5.0};

Point pt3 = pt1 + pt2;
bool b = pt1 > pt2;
bool c = pt1 == pt2;
bool d = pt1 && pt2;
```

The only operator that we may use is the assignment (=) operator

Assignment

- The assignment operator = can be used for copying a struct to another
- Example:

```
Point p1 = {1.0, 2.0}, p2;
```

```
p2 = p1;
```

is equivalent to

```
p2.x = p1.x;
p2.y = p2.y;
```

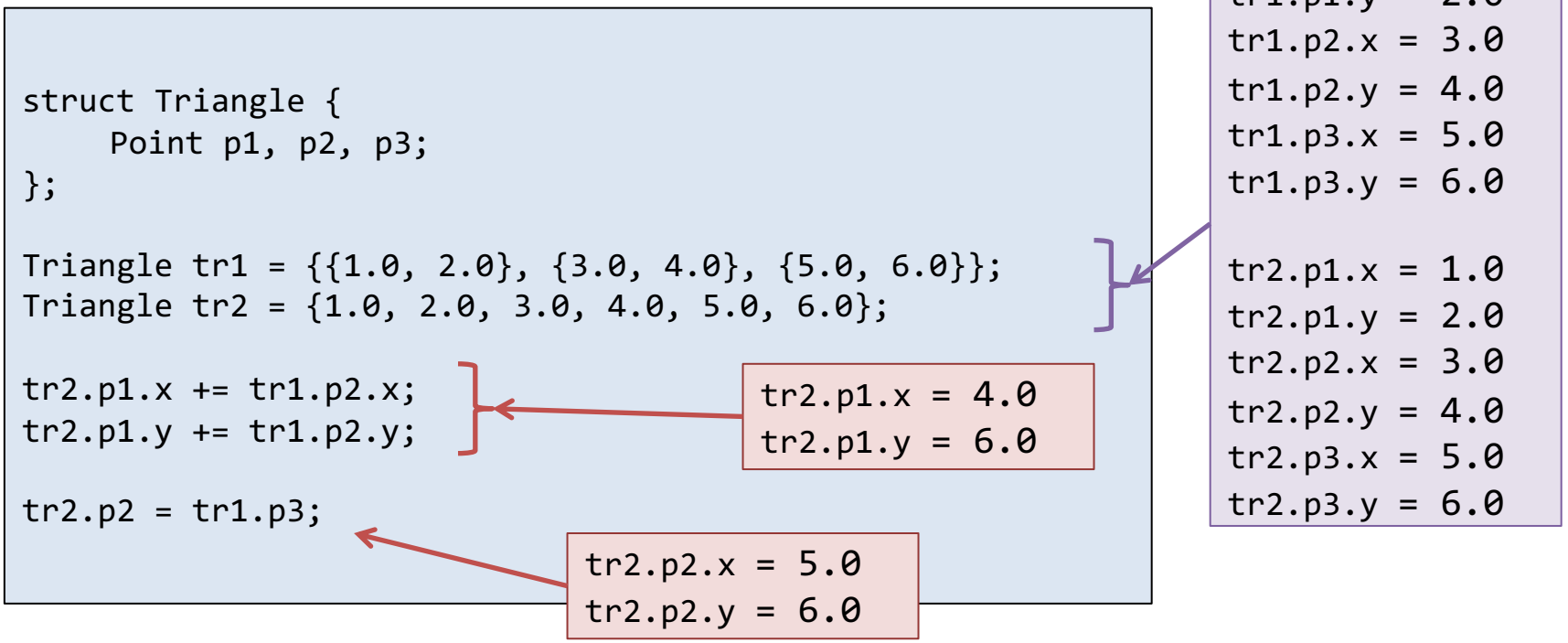
```
Point p1 = {1.0, 2.0}, p2;
p2.x = p1.y;
p2.y = p1.x;
p1 = p2;
cout << p1.x << ' ' << p1.y << endl;
```

Screen output

```
2 1
```

Nested Structures

- Structures can be nested, which means that a structure can be a member of another structure
- Examples:



Arrays of Structures

- Consider storing student records, we may use parallel arrays to store students' info and their marks :

```
const int MAX = 200;

string name[MAX];
int subclass[MAX] = {0};
int year[MAX] = {0};
int month[MAX] = {0};
int day[MAX] = {0};
double mark[MAX] = {0};
```

Elements of the same index store the info for a particular student
(e.g., `name[7]`, `subclass[7]`, `year[7]`, …)

- This is more often done using an array of `struct`, so that each element is a structure containing all the info for a student.

Parallel Arrays

```
string name[5];
int subclass[5];
int year[5];
int month[5];
int day[5];
double mark[5];
```

name	"John"	"Mary"	"Smith"	"Jordan"	"Bruce"
subclass	0	1	1	2	0
year	2014	2014	2014	2014	2014
month	10	10	10	10	11
day	28	22	29	12	1
mark	80.5	66.5	99	86.5	70.5

A record is referred to by `name[i]`, `subclass[i]`, `year[i]`, `month[i]`, `day[i]`, `mark[i]`

Array of Structures

```
struct Student_rec {
      string name;
      int subclass;
      int year;
      int month;
      int day;
      double mark;
};

Student_rec student[5];
```

student

"John"	"Mary"	"Smith"	"Jordan"	"Bruce"
0	1	1	2	0
2014	2014	2014	2014	2014
10	10	10	10	11
28	22	29	12	1
80.5	66.5	99	86.5	70.5

A record is referred to by `student[i].name`, `student[i].subclass`, `student[i].year`, `student[i].month`, `student[i].day`, `student[i].mark`

Arrays of Structures

- Student records stored in an array of struct:

```
const int MAX = 200;

struct Student_rec {
     string name;
     int subclass;
     int year;
     int month;
     int day;
     double mark;
};

Student_rec student[MAX];
```

This declares an array of size MAX, each element being a `Student_rec`.

What is the data type of each of the following?

- `student` — Array of `Student_rec`
- `student[2]` — `Student_rec`
- `student[4].year` — `int`
- `Student_rec.day` — invalid. `Student_rec` is a data type, not a variable
- `student.mark` — invalid. `student` is an array, not a struct and hence no member to access

array_structure.cpp

Arrays of Structures

- Examples:

```
// to print out the student records
for (int i = 0; i < 10; ++i) {
    cout << student[i].name << ' '
        << student[i].subclass << ' '
        << student[i].mark << endl;
}
```

```
// to copy student records
student[10] = student[5];
```

Think about this: How would you copy student records if they are stored using parallel arrays?

Take a look at array_structure.cpp
which serves the same purpose as processmarks.cpp
but using arrays of structures instead.

Structures and Functions

- Structure variables can be passed to a function either by value or by reference, and can be returned by a function like regular variables

Pass-by-value

```
// distance between two points p and q
double point_distance( Point p, Point q ) {
    double dx = p.x - q.x;
    double dy = p.y - q.y;
    return sqrt( dx * dx + dy * dy );
}
```

spoint.cpp

Compare with this:

```
// distance between two points (x1, y1), (x2, y2)
double distance( double x1, double y1,
                 double x2, double y2) {
    …
}
```

Using structure as parameters is clearer and more structural

Structures and Functions

```
// swap two points p and q
void swap( Point &p, Point &q ) {
    Point temp = p;
    p = q;
    q = temp;
}
```

Pass-by-reference

```
// get a point from user input
Point input_point() {
    double x, y;
    cin >> x >> y;
    Point p = { x, y };
    return p;
}
```

Return a structure

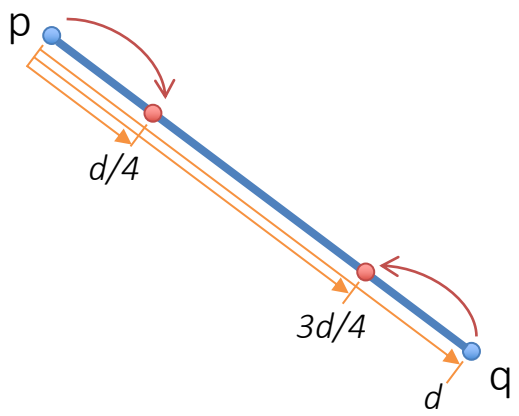
spoint.cpp

Exercise

- Add a function named **`midpoint()`** in `spoint.cpp`, which returns the mid-point of two 2D points.

```
// mid-point of two points p and q
Point midpoint( Point p, Point q ) {
    …
}
```

- Add a function named **`shrink_line()`** in `spoint.cpp`, which shrink a line defined by two endpoints as follows:




```
// shrink a line with endpoints p and q
void shrink_line( Point &p, Point &q ) {
    …
}
```

For Reference Only

Important: We expect students to be able to understand codes for class implementation, but students are not required to write code to implement a class

CLASSES

For Reference Only

Abstract Data Types

- Sometimes we would like a certain data type to be associated with specific operations.
  - Integers: +, –, *, /
  - Points: translate, distance
  - Strings: length, substring, replace
- An abstract data type (ADT) encapsulates both the data and the methods (i.e., operations) of into a package, so that users are restricted to perform only certain operations against the data inside. Also, the implementation details (how the data is stored, how the operations are carried out) of an ADT is hidden from the user (aka encapsulation or information hiding).

For Reference Only

Abstract Data Types

- To use an ADT, we only care about what can be done with them (i.e., the operations / interface), but not how they are done (i.e., the implementation).

```
string s = "I am mysterious";

cout << s.length() << endl;
cout << s.substr(0, 5) << endl;
cout << s.find("am") << endl;
```

When you use a string object, do you need to know how the string is stored internally, and how its length is determined?

As a user for the string class, we only care about what operations are available.

This is like when we use a function, we only need to know what it does by looking at its prototype, e.g.,
`double sqrt( double x);`
but we don't care about how it comes up with the result.

For Reference Only

Abstract Data Types

Consider this: When we use `struct Point`, we need to know how the coordinates are stored if we need to write a function to do anything on them.

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

```
// distance between two points p and q
double point_distance( Point p, Point q ) {
    double dx = p.x - q.x;
    double dy = p.y - q.y;
    return sqrt( dx * dx + dy * dy );
}
```

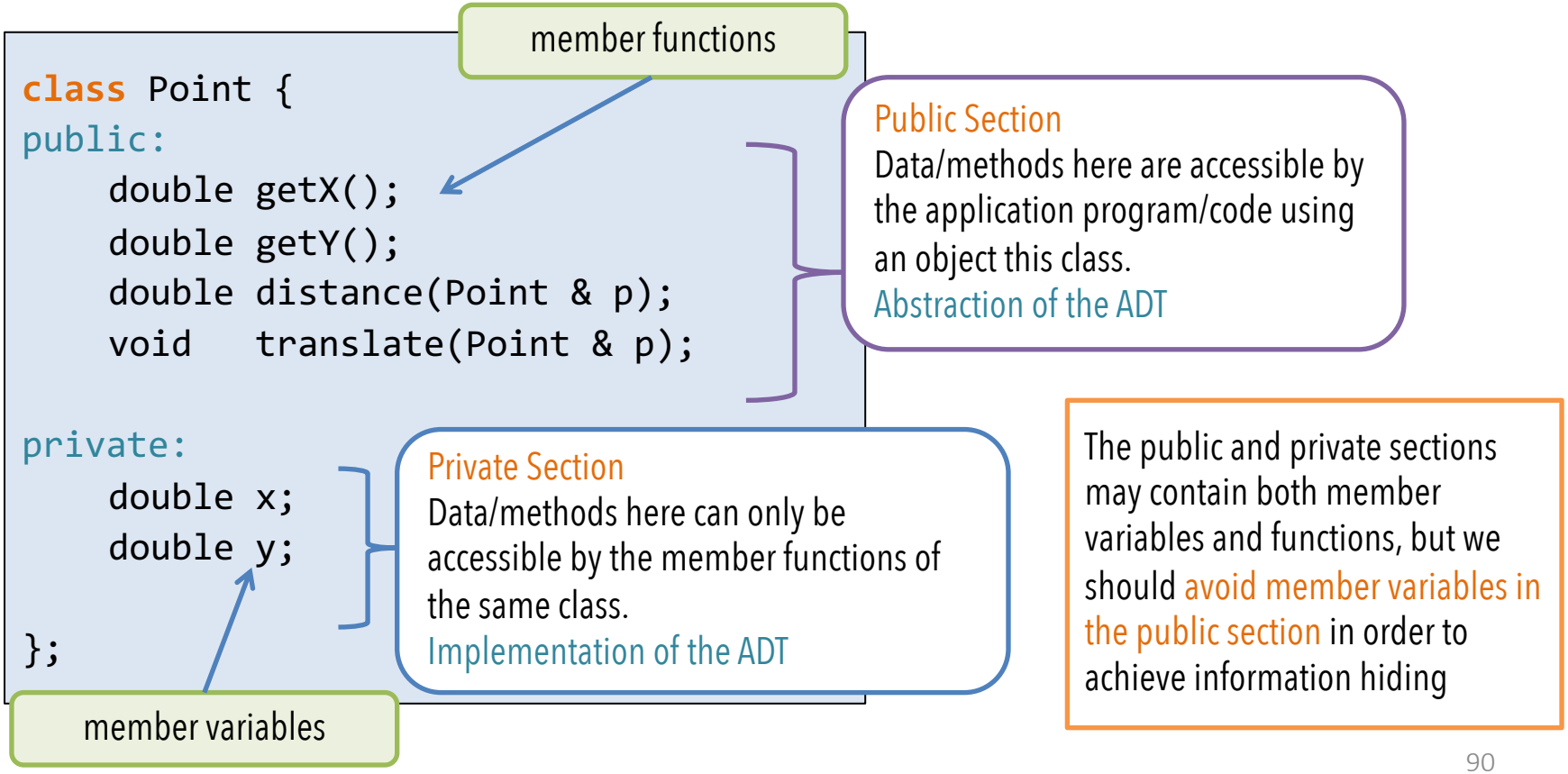
What if we later change our mind and want to use an array of 2 doubles instead to store x and y? Then any function making use of Point (e.g., point_distance()) will need to be modified.

```
struct Point {
    double v[2];
};
```

For Reference Only

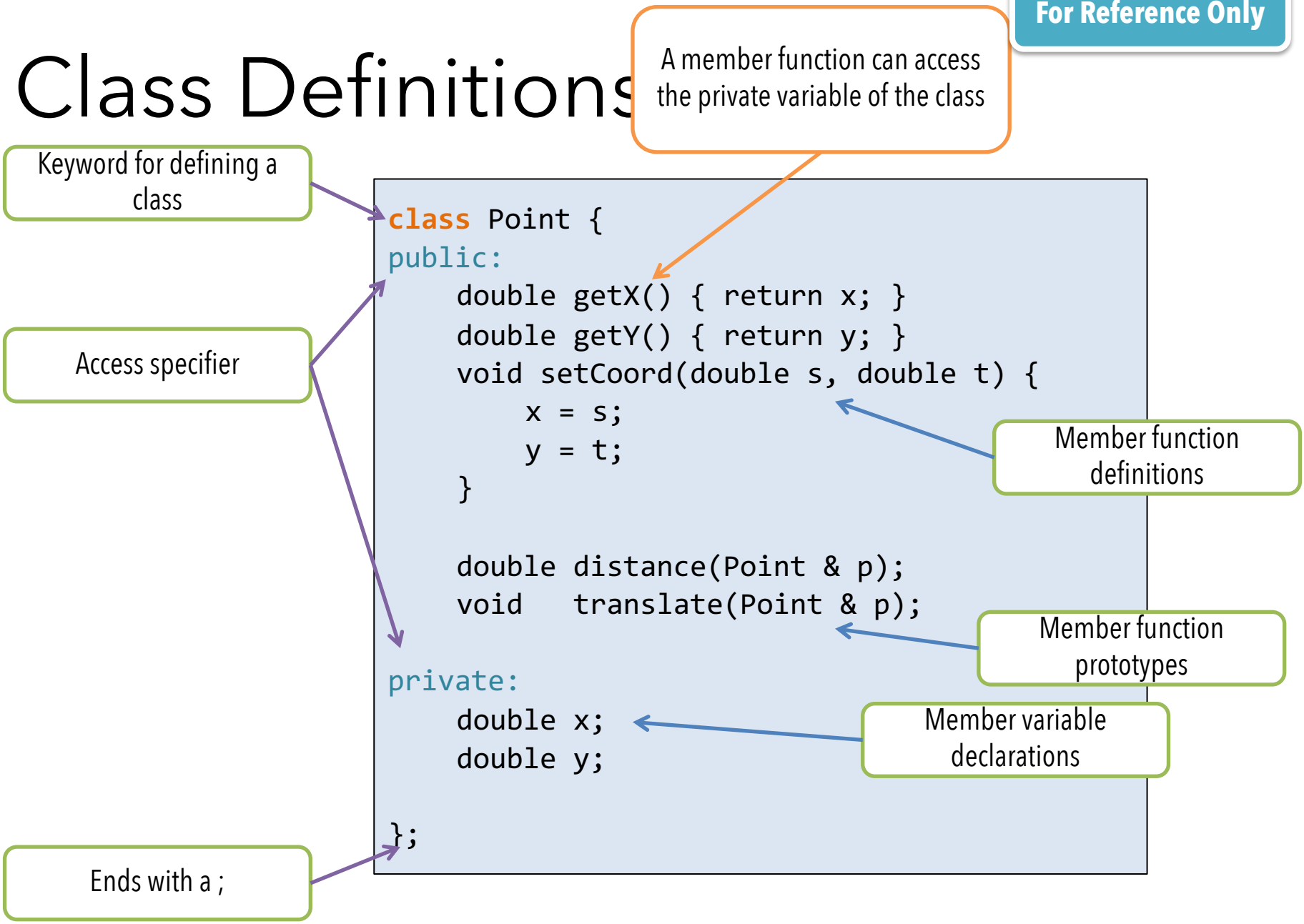
Classes

- ADTs are implemented using **classes** in C++. A class contains data (member variables) and methods (member functions) and is divided into two sections.



Class Definitions

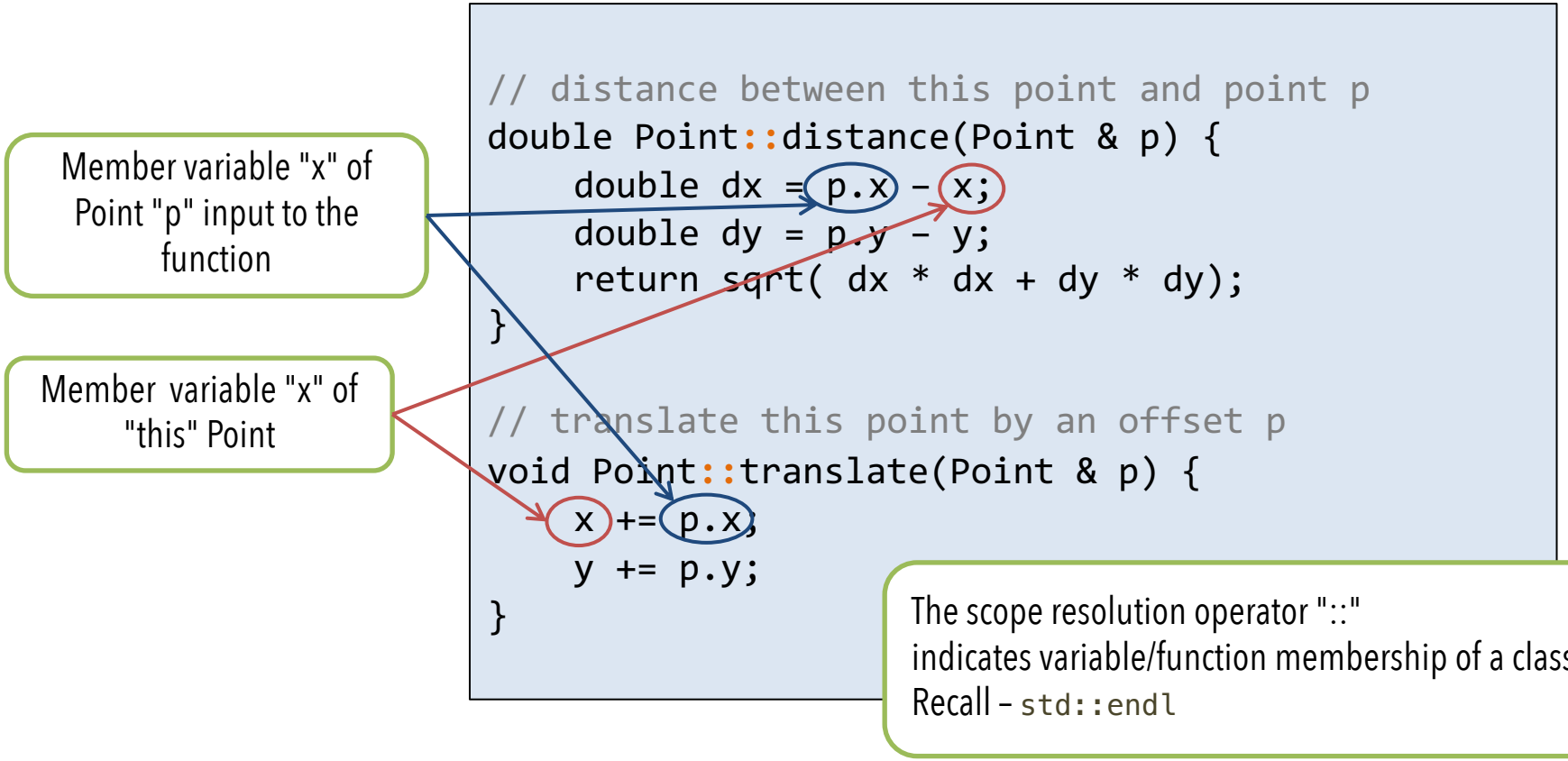
For Reference Only



For Reference Only

Member Functions

Member functions can be defined outside the class body as follows:



For Reference Only

Class Declaration

- To declare an object (variable) for a class:

```
Class_name    object_name1, object_name2, …;
```

Examples:

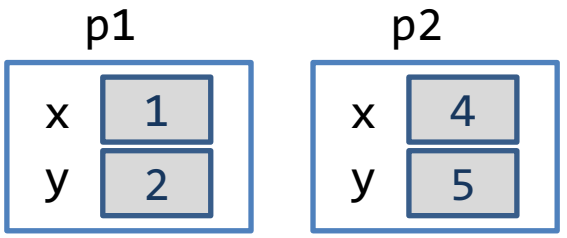
```
Point p1, p2;
string s1("abc");
```

"p1", "p2" are Point objects,
"s1" is a string object

- Each object can then retain their own values for each member variables

Examples:

```
p1.setCoord(1, 2);
p2.setCoord(4, 5);
```



For Reference Only

Multiple Files Compilation

- It is a common practice to put the codes for a class in a separate file, so that the class can be reused in another file or program.
- We also further separate the definition and implementation of a class in .cpp and .h file, respectively.

```
#include <iostream>
#include "point.h"
using namespace std;

int main()
{
      …
      return 0;
}
```

main.cpp

Main program

```
class Point
{
public:
      …

private:
      …
};
```

point.h

Class interface

```
double Point::distance(Point & p) {
      …
}

void Point::translate(Point & p)
{
      …
}
```

point.cpp

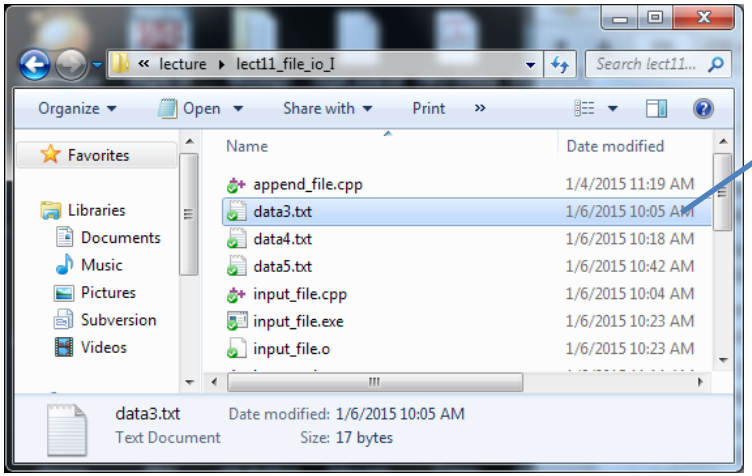
Class implementation

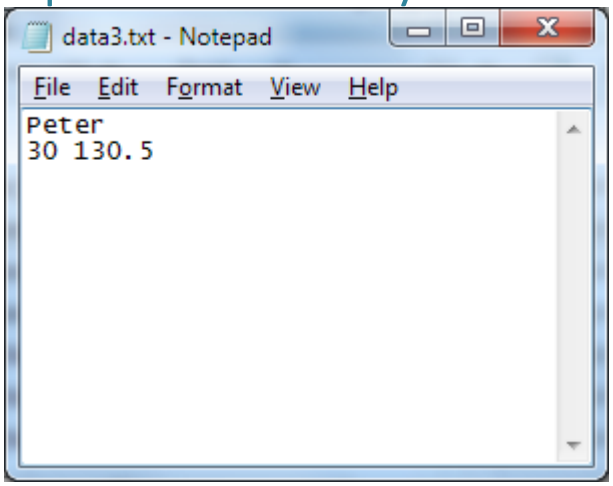
Any other program that wants to use Point can just include "**point.h**".

File I/O

File Input/Output

- Files are used for storing data permanently.





Contents of "data3.txt"

- C++ simply views a file as a sequence of bytes:

'P'	'e'	't'	'e'	'r'	'\n'	'3'	'0'	' '	'1'	'3'	'0'	'.'	'5'	eof

A file in the file system named "data3.txt"

end of file marker

Streams

- C++ uses a convenient abstraction called streams to perform input and output operations in sequential media, e.g.,
  - cout is a stream object for sending output to the screen
  - cin is a stream object for taking input from keyboard
- C++ provides two classes, namely ofstream and ifstream, for writing and reading data to and from files
- To use the classes ofstream and ifstream, simply include the header file fstream, i.e.,

```
#include <fstream>
```

WRITE TO FILE

Output File Stream

- A basic example for creating and writing to a file

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <string>
using namespace std;

int main()
{
   ofstream fout;
   fout.open("data1.txt");

   if ( fout.fail() ) {
      cout << "Error in file opening!"
           << endl;
      exit(1);
   }
```

```
   string name = "Peter";
   int age = 30;
   double weight =

   fout << name << " " << age << " "
        << weight << endl;
   fout.close();

   return 0;
}
```

Include the file stream header file

Create an **ofstream** (output file stream) object and connect it to an **external file** named "data1.txt"

These two statements can be replaced by:
ofstream fout ("data1.txt");

data1.txt

output_file.cpp

Output File Stream

- A basic example for creating and writing to a file

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <string>
using namespace std;

int main()
{
   ofstream fout;
   fout.open("data1.txt");

   if ( fout.fail() ) {
      cout << "Error in file opening!"
           << endl;
      exit(1);
   }
   string name = "Peter";
   int age =
   double w

   fout << name << " " << age << " "
        << weight << endl;
   fout.close();

   return 0;
}
```

This **if** block serves to exit the program if unable to create file.

Function **exit** forces a program to terminate immediately, and is often used to terminate a program when an error is detected in the input or if a file to be processed by the program cannot be opened.

output_file.cpp

Output File Stream

- A basic example for creating and writing to a file

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

int main()
{
   ofstream fout;
   fout.open("data1.txt");

   if ( fout.fail() ) {
      cout << "Error in file openi
           << endl;
      exit(1);
   }

   string name = "Peter";
   int age = 30;
   double weight = 130.5;

   fout << name << " " << age << " "
        << weight << endl;
   fout.close();

   return 0;
}
```

Write to the file stream **fout** using the insertion operator <<
(just as what we do with cout)

Finally disconnects the file stream **fout** from the external file

output_file.cpp

data1.txt

```
Peter 30 130.5\n
eof
```

Summary

Steps for Creating and Writing to a File

1. Declare an output stream variable.

```
ofstream fout;
```

2. Open the file

```
fout.open("data.txt");
```

3. Check if there is any error in opening the file

```
if (fout.fail())
```

4. Use the insertion operator **<<** to write to file

```
fout << "12345";
```

5. Close the file

```
fout.close();
```

```
string filename = "data.txt";
fout.open(filename.c_str());
```

if the file name is stored as string

Appending Data to a File

- When opening a file for output using the member function **open()**, a new file will be created if the file does not already exist, otherwise the content of the existing file will be erased
- To keep the content of the existing file and append new data to it, supply the constant value ios::app as a second argument to the member function open(), e.g.,

```
fout.open("data2.txt", ios::app)
```

Appending Data to a File

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <string>
using namespace std;

int main()
{
   ofstream fout;
   fout.open("data2.txt", ios::app);

   if (fout.fail()) {
      cout << "Error in file opening!"
         << endl;
      exit(1);
   }
   string name = "John";
   int age = 25;
   double weight = 129.3;

   fout << name << " " << age << " "
      << weight << endl;
   fout.close();

   return 0;
}
```

append_file.cpp

data2.txt
(before executing the program)

```
Peter 30 130.5\n
eof
```

data2.txt
(after executing the program)

```
Peter 30 130.5\n
John 25 129.3\n
eof
```

READ FROM FILE

Input File Stream

- A basic example for reading from an existing file

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <string>
using namespace std;

int main()
{
    char filename[80] = "data3.txt";
    ifstream fin;
    fin.open(filename);

    if

    }
```

```
    string name;
    int age
    double

                >> weight;


                << age << ", "

}
```

Include the file stream header file

Create an **ifstream** (input file stream) object and connect it to an **external file** named "data3.txt"

These few statements can be replaced by:
ifstream fin ("data3.txt");

Since the **open()** function accepts only a C-string as the input parameter, if the file name is stored in a string class, we will need to write:

string filename = "data3.txt"
ifstream fin(filename.c_str());

input_file.cpp

data3.txt

```
Peter\n
30 130.5\n
eof
```

Input File Stream

- A basic example for reading from an existing file

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <string>
using namespace std;

int main()
{
   char filename[80] = "data3.txt";
   ifstream fin;
   fin.open(filename);

   if ( fin.fail() ){
      cout << "Error in file opening!"
         << endl;
      exit(1);
   }
   string name;
   int age;
   double weight;

   fin >> name >> age >> weight;
   fin.close();

                          " << age << ", "
                          dl;
   return 0;
}
```

Exit the program if the file does not exist

input_file.cpp

data3.txt

```
Peter\n
30 130.5\n
eof
```

Input File Stream

- A basic example for reading from an existing file

```
#include <iostream>
#include <fstream>
int main()
{
   ifstream fin;
   fin.open("data3.txt");
   if ( fin.fail() ){
      cout << "Error in file opening!"
         << endl;
      exit(1);
   }
```

Read from the file stream **fin** using the extraction operator >> (just as what we do with cin)

Finally disconnects the file stream **fin** from the external file

input_file.cpp

```
   string name;
   int age;
   double weight;

   fin >> name >> age >> weight;
   fin.close();

   cout << name << ", " << age << ", "
      << weight << endl;
   return 0;
}
```

data3.txt

```
Peter\n
30 130.5\n
eof
```

Screen output

```
Peter, 30, 130.5
```

Summary

Steps for Reading Input from a File

1. Declare an **ifstream** object.

```
ifstream fin;
```

2. Open the file

```
fin.open("data.txt");
```

3. Check if there is any error in opening the file

```
if (fin.fail())
```

4. Read data from file using the extraction operator **>>**

```
fin >> x;
```

5. Close the file

```
fin.close();
```

Reading until End of File (EOF)

- Very often, data have to be extracted sequentially from an input file until the end of file (eof) has been reached (because we don't know the length of a file in advance)
- This can be done by using a while loop as follows:

```
while (fin >> x)
{
    …
}
```

- The return value of the expression fin >> x:
  - A nonzero (true) value indicates a datum has been read successfully
  - A zero (false) value indicates the eof has been reached and no datum has been read

Reading until End of File (EOF)

- Example

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <string>
using namespace std;

int main()
{
   ifstream fin;
   fin.open("data4.txt");
   if (fin.fail()) {
      cout << "Error in file opening!"
         << endl;
      exit(1);
   }
   double x, sum = 0;

   while (fin >> x) {
      sum += x;
   }

   fin.close();
   cout << "Total = " << sum
        << endl;
   return 0;
}
```

Read and sum until end of file

read_till_eof.cpp

data4.txt

```
20.0 40.0 60.0 eof
```

Screen output

```
Total = 120
```

Reading Lines From a File

- Sometimes, data in a file may need to be processed in a line by line manner, e.g., each line stores the record of one person
- The library function getline() can be used to read in a line from an input file stream object and store it as a string object, e.g.,

```
getline(fin, str);
```

fin is an input file stream object and str is a string object (both are call-by-reference parameters)

- Similarly, the return value of getline() can be used to check if the eof has been reached
  - A nonzero (true) value indicates a line has been read successfully
  - A zero (false) value indicates the eof has been reached and no line has been read

Reading Lines From a File

- Example:

```
#include <iostream>
#include <fstream>
#include <cstdlib>
#include <string>
using namespace std;

int main()
{
   ifstream fin;
   fin.open("data5.txt");
   if (fin.fail()) {
      cout << "Error in file opening!"
         << endl;
      exit(1);
   }
   string line;

   while ( getline(fin, line) ) {
      cout << line << endl;
   }

   fin.close();
   return 0;
}
```

read_line_file.cpp

data5.txt

```
Peter 30 130.5\n
John 129.3\n
eof
```

Screen output

```
Peter 30 130.5
John 129.3
```

Exercise

- Write a program **copyfile.cpp** that prompts the user for a file name of a text file, reads the file and writes its content to a new file. This essentially copies an existing file to another file.

Input String Stream

- C++ also provides the class istringstream for extracting data from a string. To use this class, simply include the header file <sstream>, i.e.,

```
#include <sstream>
```

- An input string stream object can be declared using the class name istringstream and initialized with a string object as follows

```
string str;
istringstream iss(str);
```

- Data can then be extracted from the input string stream using the extraction operator >>

```
int age;
iss >> age;
```

Input String Stream

- Similarly, data can be extracted sequentially from the stream until the end of string has been reached by checking the return value of the expression

```
input_string_stream >> variable
```

  - A nonzero (true) value indicates a datum has been read successfully
  - A zero (false) value indicates the end of string has been reached and no datum has been read

Input String Stream

- Example

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

int main()
{
   string line=" apple orange banana ", word;

   istringstream line_in(line);

   while ( line_in >> word ) {
      cout << "\"" << word << "\""
         << endl;
   }

   return 0;
}
```

Screen output

```
"apple"
"orange"
"banana"
```

input_string_stream.cpp

Stream Output Formatting

- Sometimes you may want to have the output from your program to be displayed (on screen) or stored (in file) in a specific format
  - Floating-point numbers: **0.00001** or **1e-5**? **15** or **15.000**?
  - Formatted tabular output:

```
Peter   30   130.5
John     6   129.3
Mary    18    34.5
```

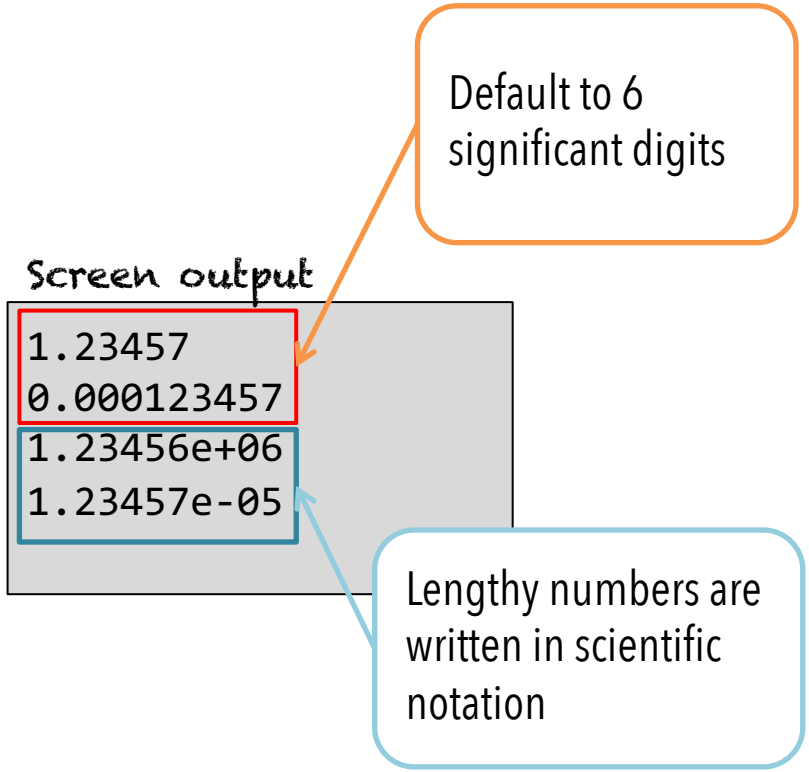
How to set the width of each column?
How to set the column alignment?

- We may use the output manipulators to format the output. We've come across some examples:
  - **endl**, to move the insertion point to the beginning of the next line
  - **setw**, to set the width of the column for the next output value

Default floating-point notation

- Example

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

int main()
{
    double a = 1.2345678;
    double b = 0.00012345678;
    double c = 1234567.8;
    double d = 0.000012345678;

    cout << a << endl << b << endl
        << c << endl << d << endl;
    return 0;
}
```



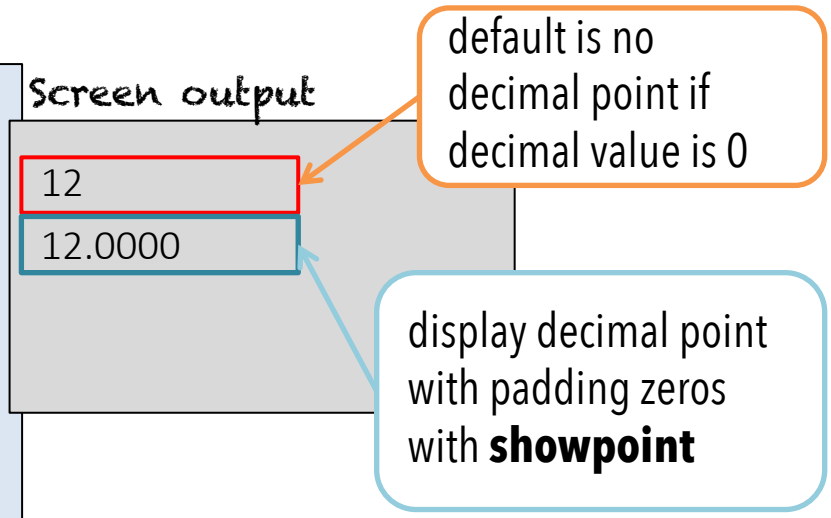
default_float.cpp

showpoint Manipulator

- Example

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

int main()
{
    double e = 12.0;

    cout << e << endl;
    cout << showpoint << e << endl;

    return 0;
}
```

default_float.cpp

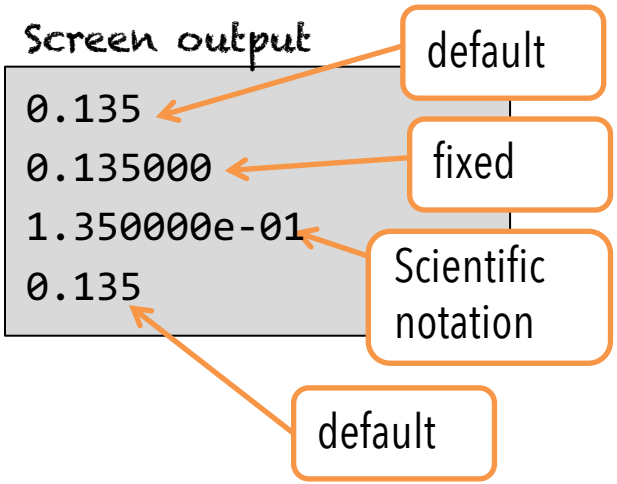


can be unset with the **noshowpoint** manipulator

fixed / **scientific** Manipulators

- **fixed** to write floating-point numbers as fixed decimal
- **scientific** to output floating-point numbers in scientific notation

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

int main()
{
    double f = 0.135;
    cout << f << endl;
    cout << fixed << f << endl;
    cout << scientific << f << endl;

    cout.unsetf(ios_base::floatfield);
    cout << f << endl;
    return 0;
}
```



manipulator_fixed.cpp

setprecision Manipulator

- With the default floating-point notation, **setprecision** specifies the maximum number of meaningful digits before and after the decimal point.

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

int main()
{
    double a = 1.2345678;
    double b = 1234567.8;
    cout << a << '\n' << b << "\n\n";

    cout << setprecision(2);
    cout << a << '\n' << b << '\n';

    return 0;
}
```

Screen output

```
1.23457
1.23457e+006

1.2
1.2e+006
```

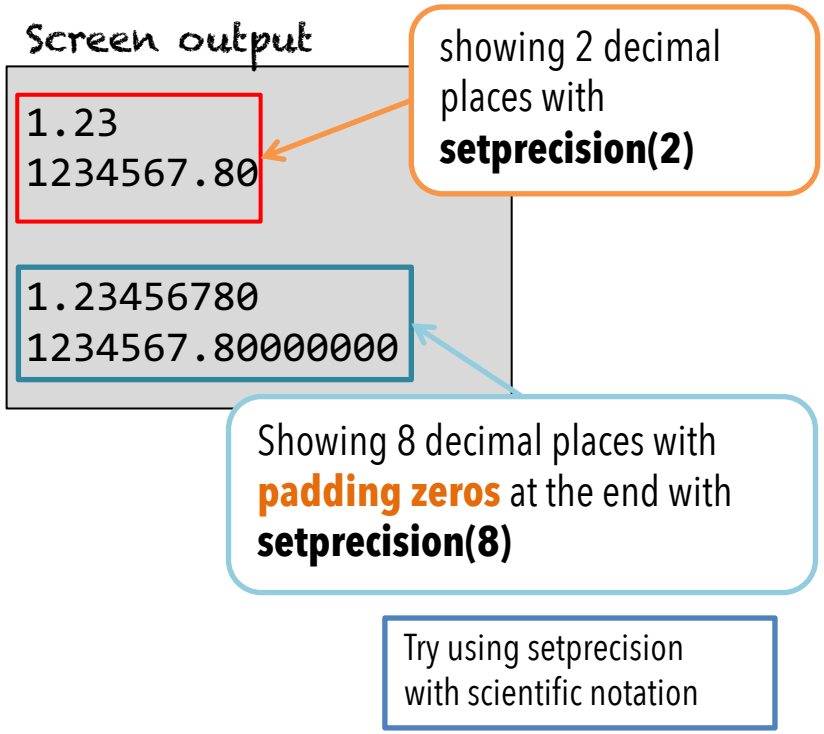
showing 2 significant digits with **setprecision(2)**

manipulator_setprecision.cpp

setprecision Manipulator

- With the fixed or scientific notation, **setprecision** specifies the exact number of digits after the decimal point. By default, 6 decimal places are used.

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

int main()
{
    double a = 1.2345678;
    double b = 1234567.8;

    cout << fixed << setprecision(2);
    cout << a << '\n' << b << "\n\n";

    cout << setprecision(8);
    cout << a << '\n' << b << '\n';
    return 0;
}
```



manipulator_setprecision.cpp

setw Manipulator

- Use setw to output a string or a number in a specific number of columns (the output is right-justified).

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

int main()
{
    int x = 12;
    string a = "Hello";
    double b = 34.567;

    cout << fixed << setprecision(2);
    cout << "12345678901234567890\n";

    cout << setw(5) << x << setw(8) << a;
    cout << setw(6) << b << endl;

    return 0;
}
```

Screen output

```
12345678901234567890
   12   Hello 34.57
```

5 cols 8 cols 6 cols

For those manipulators that accept parameters such as setw(x), include the **<iomanip>** header; otherwise for those manipulator without parameters such as fixed, include the **<iostream>** header

manipulator_setw.cpp

setfill Manipulator

- With setw, if the specified number of columns > the required number of columns, the unused columns are filled with spaces. We may use setfill to fill the unused columns with other characters.

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

int main()
{
    int x = 12;
    string a = "Hello";
    double b = 34.567;

    cout << fixed << setprecision(2);
    cout << "12345678901234567890\n";
    cout << setfill('*');
    cout << setw(5) << x << setw(8) << a;
    cout << setw(6) << b << endl;

    return 0;
}
```

Screen output

```
12345678901234567890
***12***Hello*34.57
```

manipulator_setw.cpp

left / **right** Manipulators

- With setw, the default output is right-justified within a column. Use the **left** and **right** manipulators to set the output to be left-justified or right-justified, respectively.

```
…
cout << "123456789012345678901\n";
cout << setfill('-');

cout << left;
cout << setw(5) << x << setw(8) << a;
cout << setw(6) << b << endl;

cout << right;
cout << setw(5) << x << setw(8) << a;
cout << setw(6) << b << endl;
…
```

manipulator_setw.cpp

Screen output

```
12345678901234567890
12---Hello---34.57-
---12---Hello-34.57
```

`left` and `right` are defined in <iostream>

Further References on File I/O

- C++ Language Tutorial: Input/Output with files
  http://www.cplusplus.com/doc/tutorial/files/
- C++ Library Reference: ifstream class
  http://www.cplusplus.com/reference/fstream/ifstream/
- C++ Library Reference: istringstream class
  http://www.cplusplus.com/reference/sstream/istringstream/
- C++ Library Reference: ofstream class
  http://www.cplusplus.com/reference/fstream/ofstream/
- C++ Library Reference: ofstream class
  http://www.cplusplus.com/reference/library/manipulators/

PROBLEMS

Problem 1

Write a program that will read 8 characters into an array and write the letters back to the screen in reverse order. For example, if the input is abcdefgh, then the output should be hgfedcba.

Problem 2

(The Sieve of Eratosthenes) A prime integer is any integer that is evenly divisible only by itself and 1. The Sieve of Eratosthenes is a method of finding prime numbers. It operates as follows:

A. Create an array with all elements initialized to true (what is the data type of this array?). Array elements with prime subscripts will remain true throughout the program execution. All other array elements will eventually be set to false. You'll ignore the first two elements with indexes 0 and 1 in this question.
B. Starting with array index 2, every time an array element is found whose value is true, loop through the remainder of the array and set to false every element whose index is a multiple of the index for the element with value 1. For instance, for array index 2, all elements beyond index 2 in the array that are multiples of 2 will be set to false (indexes 4, 6, 8, 10, etc.); for array index 3, all elements beyond 3 in the array that are multiples of 3 will be set to false (indexes 6, 9, 12, 15, etc.); and so on.

When this process is complete, the array elements that are still set to true indicate that the index is a prime number. These indexes can then be printed. Write a program that uses an array of 1000 elements to determine and print the prime numbers between 2 and 999.

Problem 3

Write a function named swapFrontBack that takes as input an array of integers and an integer that specifies how many entries are in the array. The function should swap the first element in the array with the last element in the array. The function should check if the array is empty to prevent errors. Test your function with arrays of different length and with varying front and back numbers.

Problem 4

Write a program that converts a two-digit number entered by the user to words. The program takes a maximum of two-digit numbers only. For instance, if the user enters 2 it should write “two”, and if 34 is entered, it should write “thirty-four”. The minimum number that can be entered is 1 and the maximum number is 99. Your program should make use of arrays that store the fundamental numbers in words, and use modulus and integer division to do the required conversion.

Problem 5

Write a function to copy the contents of an array to another array. (You may pass two arrays, a source array and a destination array, to the function as arguments.)

Problem 6

Write a function isPalindrome that determines if a **char array** is a palindrome. You may assume that the char array is filled with chars from 'a' to 'z'. A palindrome is one which reads the same from the beginning and from the end. Example, "abcbc", "noon", "kayak" are palindromes.

Problem 7

Create a text file and save it on your computer. Here is an example.

twocities.txt

```
There were a king with a large jaw and a queen with a plain face, on the
throne of England; there were a king with a large jaw and a queen with a fair
face, on the throne of France. In both countries it was clearer than crystal
to the lords of the State preserves of loaves and fishes, that things in
general were settled for ever.

```

Write a C++ program that reads in the content of the file and then outputs only every second word on the screen. Your program should output the following for the given text file.

twocities.txt

```
were king a jaw a with plain on throne England; were king a jaw a with fair
on throne France. both it clearer crystal the of State of and that in were for

```

Problem 8

Write a program that will search a file of numbers of type int and write the largest and the smallest numbers to the end of that file. The file contains nothing but numbers of type int separated by spaces or line breaks.

Problem 9

Write a program that will read in a file containing nothing but numbers of type int separated by spaces or line breaks. Sort the numbers and write them into a new file.

Problem 10

Fill in the body of the `equal` function below. The function will return true if the member variables of the structure contain the same data and false otherwise.

lab08-q4.cpp

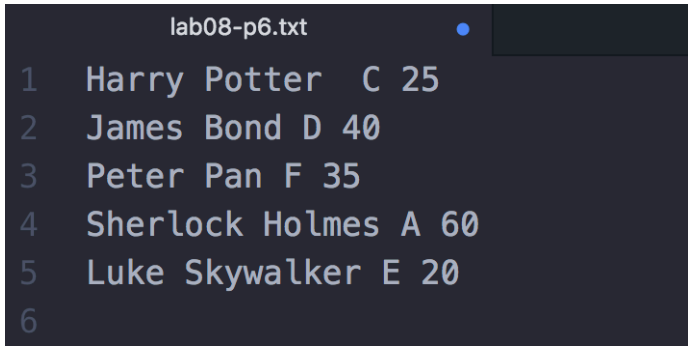
```cpp
#include <iostream>
using namespace std;

struct Entry {
    string firstName;
    string lastName;
    char dorm;
    int age;
};

// write the function definition for equal() here


int main() {
    Entry e1 = {"Harry", "Potter", 'C', 25};
    Entry e2 = {"James", "Bond", 'D', 40};

    if (equal(e1, e2))
        cout << "same" << endl;
    else
        cout << "different" << endl;
}

```

Problem 11

Rewrite the function of Problem 10 such that it takes just a single parameter. Move the function inside the struct definition.

Problem 12

Read a text file that consists of a list of Entry. Here is an example:



Save the entries as an array of type **`Entry`**. Sort the list by age and output the result on the screen. You may assume that there are at most 100 entries in the text file.

Optional.

For those who would like to challenge yourselves.
Even for those of you who are beginners in C++ programming, it's highly recommended for you to take a look at these problems and try to tackle them as well.

You are welcome to discuss these problems in the Moodle forum.

CHALLENGES

Challenge 1

Using similar idea of the sieve table that you have implemented in Problem 2 above, write a program that determines the prime factorization of an input integer. For example, given the input number 24, your program should output 2x2x2x3 and for the input number 30, the output should be 2x3x5. Hint: You may want to store integer values instead of Boolean values in the sieve table.

Challenge 2

A playing card consists of a suit (A, B, C, D) and a number (1 to 13), e.g. D12. Two cards are said to be a pair when they have the same number. Construct a program to read 10 playing cards from the user, and then output the number of pairs. You can assume the input playing cards are always valid.

SAMPLE INPUT	SAMPLE OUTPUT
A13 B5 D6 C5 B8 A6 C4 B10 D5 C6	2 Pairs
A2 A1 B2 B1 C2 C1 D2 D3 D5 B5	4 Pairs
B6 B9 A9 C9 D12 D6 A6 C6 A12 D9	5 Pairs

Challenge 3

Implement a function `erase` to remove part of characters in a character array.

The function header should be:

```
void erase(char str[], int pos, int len)
```

which removes a part of characters stored in `str`, starting from position, and with length `len`.

Challenge 3 (Continue)

For example, if `text[]` stores "Happy B-day":

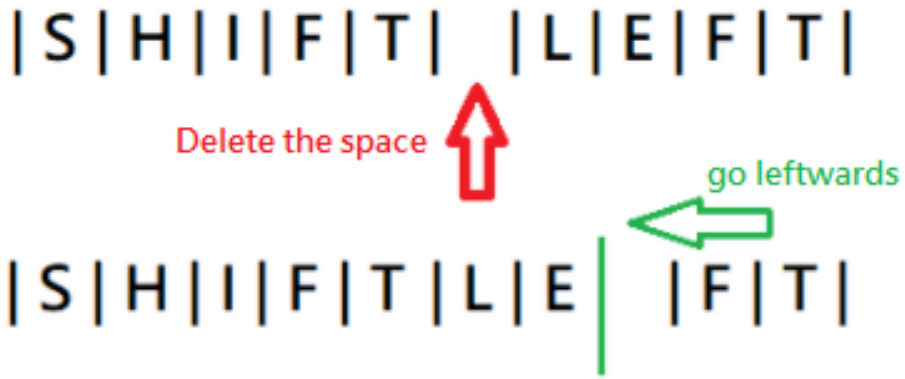
'H'	'a'	'p'	'p'	'y'	''	'B'	'-'	'd'	'a'	'y'	'\0'

then after calling `erase(text, 6, 2)`, `text[]` should store "Happy day":

'H'	'a'	'p'	'p'	'y'	''	'd'	'a'	'y'	'\0		

Challenge 3 (Continue)

To erase a character from a array, you may use a for loop to shift some portion of the string leftwards, and reduce the length of the character array by one.



Implement the erase function and also write a program with the main body to test the function.