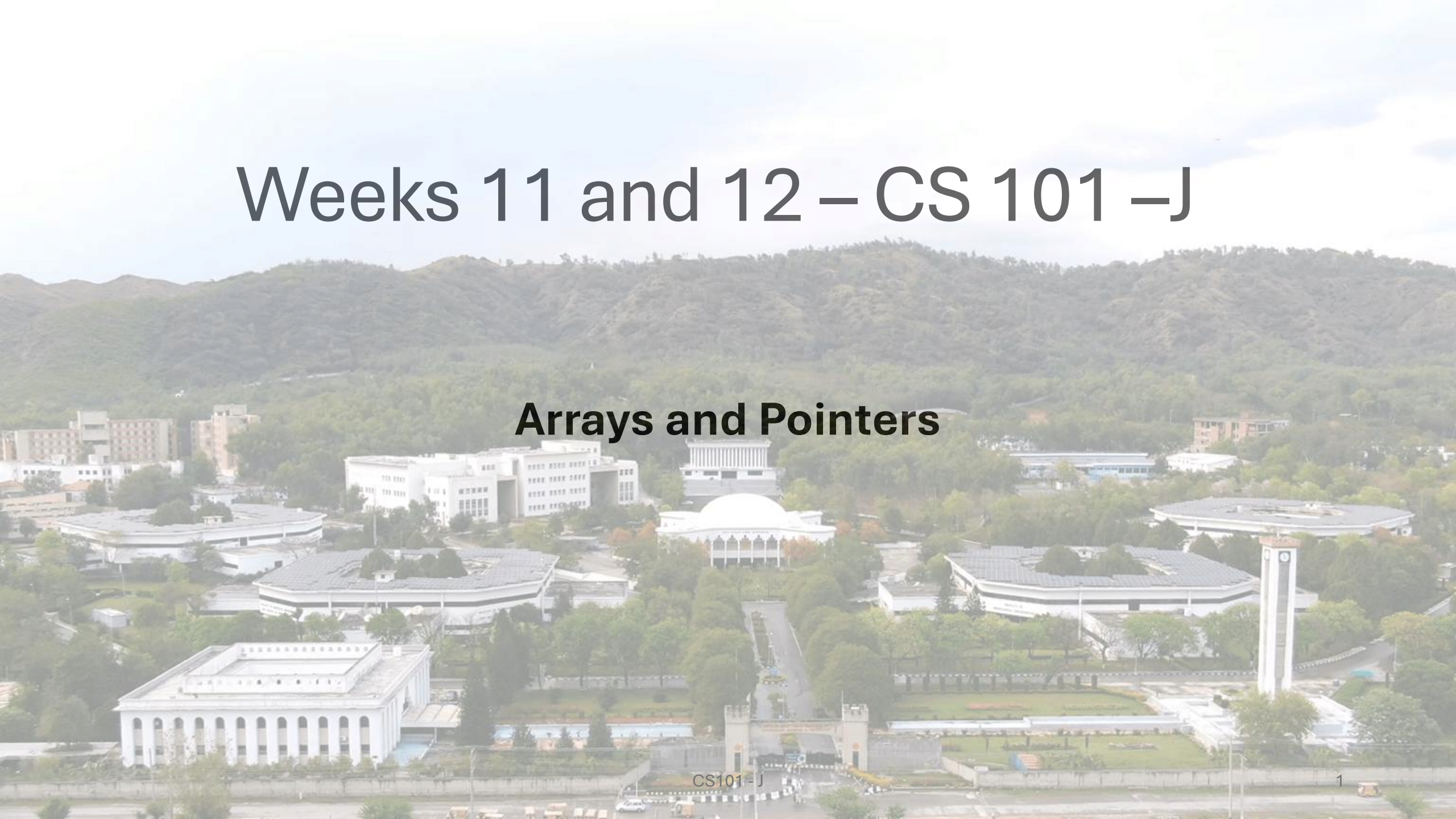
Weeks 11 and 12 – CS 101 –J



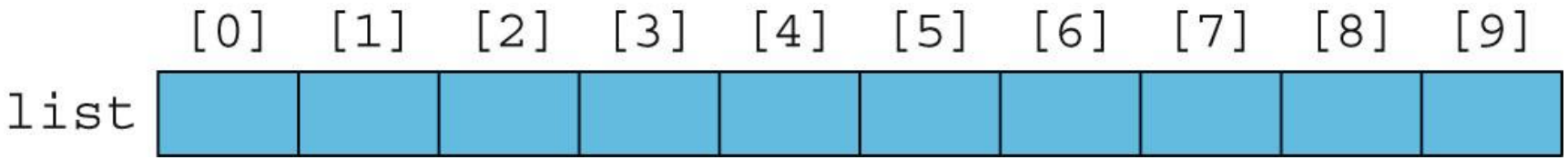
Arrays and Pointers

Operations on Arrays

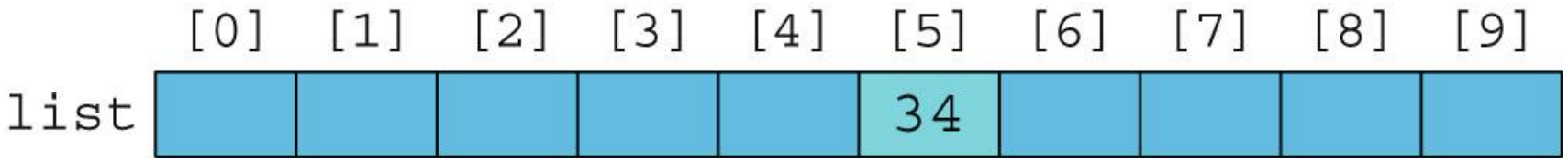
- Basic operations on an array:
  - Initializing
  - Inputting data
  - Outputting data stored in an array
  - Finding the largest and/or smallest element
- Each operation requires ability to step through elements of the array

Array Initialization

```
int list[10];
```

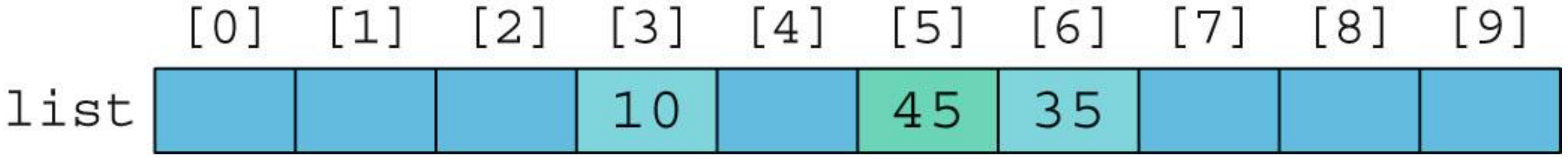



```
list[5] = 34;
```



Array Element Assignment and Operations

```
list[3] = 10;
list[6] = 35;
list[5] = list[3] + list[6];
```



Input Elements in an Array

- Given the declaration:

```
int a[100];  //array of size 100
int i;
```

- Use a **for** loop to access array elements:

```
for (i = 0; i < 100; i++)
     cin >> a[i];
```

Array index Out of Bounds

- Index of an array is in bounds if the index is **>=0** and **<= `ARRAY_SIZE-1`**
  - Otherwise, the index is out of bounds

- In C++, there is no guard against indices that are out of bounds

More Examples of Initialization

- The statement:

```
int b[10] = {0};
```

  - Declares an array of 10 components and initializes all of them to zero

- The statement:

```
int c[10] = {8, 5, 12};
```

  - Declares an array of 10 components and initializes `c[0]` to 8, `c[1]` to 5, `c[2]` to 12
  - All other components are initialized to **0**

Operations on Arrays

- Remember: We cannot directly ADD arrays in one operation – this operation **has to be element wise**

```
int d[] = {8, 5, 12};
int e[] = {10,20,30};
int f[]= d + e;              // Not allowed

for (int i=0; i <3;  i++)     // Allowed
      f[i]=d[i]+f[i];         // Allowed
```

Arrays as Parameters to Functions

- Arrays are passed by reference only
- The usage of word **const** prevents a function from changing the actual parameters of an array

- Execute wk11s9.cpp and understand the following points:
  - A function is not returning anything (fun1) – still it is modifying 5 values in the main function
  - A function (fun2) has been prevented from modifying the values passed to it be reference

Base Address of an Array

- Base address of an array: address (memory location) of the first array component

- Example:
  - If **`A`** is an array, its base address is the address of **`A[0]`**

- When an array is passed as a parameter, the base address of the actual array is passed to it

- Understand and execute wk11s10.cpp

Function Return Type

- C++ does not allow functions to return a value of type array

- However, when we study pointers, then we will study about pointer pointing to arrays, and a pointer being returned from a function.

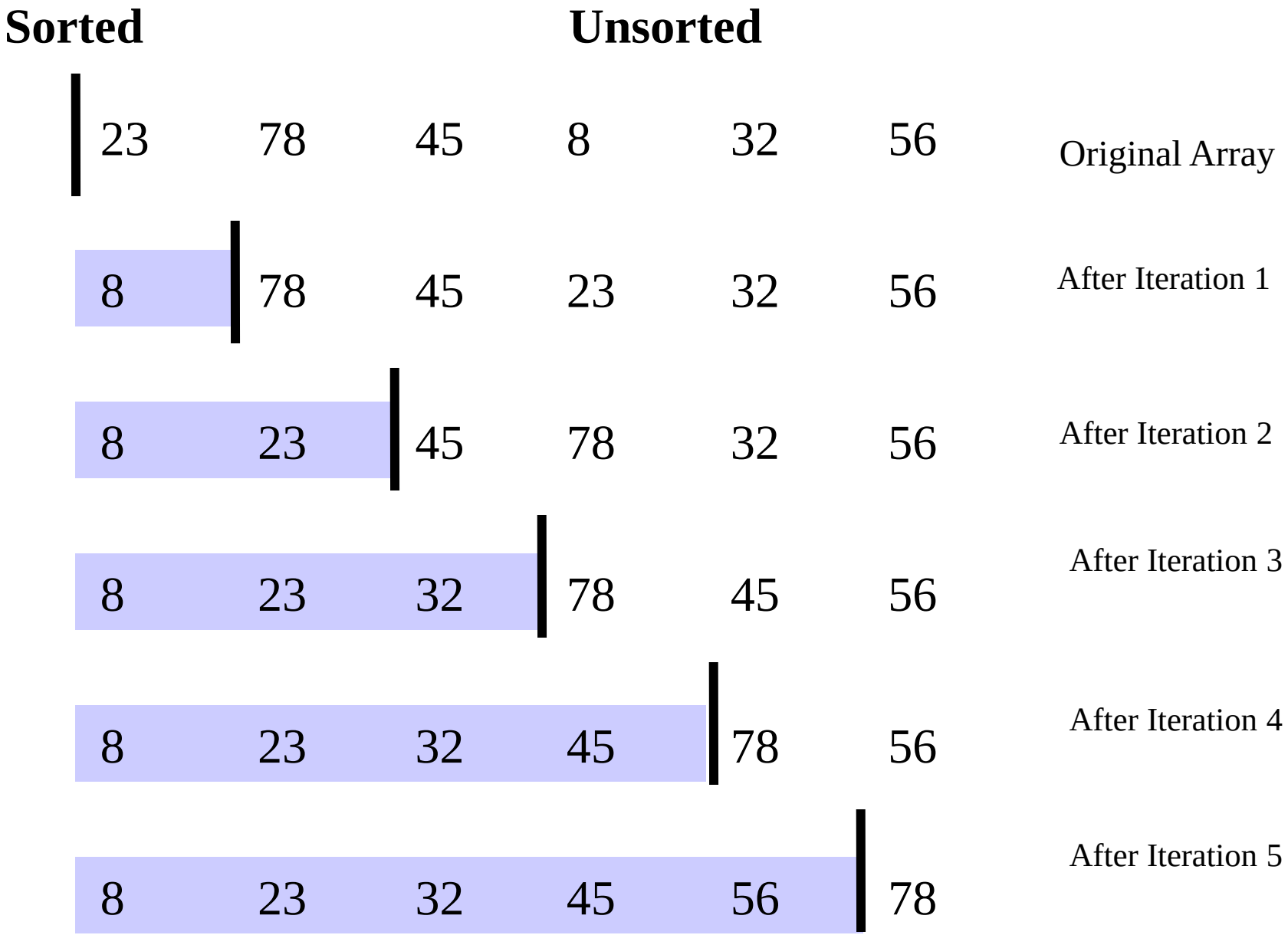
Searching and Sorting Arrays

- Sequential search (or linear search):
  - Searching a list for a given item, starting from the first array element
  - Compare each element in the array with value being searched for
  - Continue the search until item is found or no more data is left in the array

Sorting Arrays

- Selection sort: rearrange the array by selecting an element and moving it to its proper position

- Steps:
  - Divide/Partition the array into unsorted and sorted part (initially, sorted part is empty)
  - Find the smallest element in the unsorted portion of the array
  - Swap it with the current position in sorted part
  - Start again with the rest of the elements in un-sorted part


Sorted
Unsorted
23 78 45 8 32 56
Original Array
8 78 45 23 32 56
After Iteration 1
8 23 45 78 32 56
After Iteration 2
8 23 32 78 45 56
After Iteration 3
8 23 32 45 78 56
After Iteration 4
8 23 32 45 56 78
After Iteration 5

Sorting Arrays

- Now, if we understand the algorithm of Selection Sort, lets translate it into C++ language
- How many variables we need?
  - Let us call the size of array as n (n=6 in case of previous slide)
  - We need a variable to tell us about the location of minimum value in the array (position of min or pos_of_min)
  - We need a variable starting from 0 (initial index of sorted part), and going till n-1 (Remember: sorted part has no element in the beginning, then 1 element in iteration 1, 2 elements in iteration 2, and so on…) – Call this as variable i
  - In which part of array we are supposed to locate pos_of_min? Start with i+1 and go till end

Understand wk11s16.cpp

- At each iteration, understand the values of elements in Array, the role of variable i and j, and pos_of_min

Array Sorting

- There are many sorting techniques for array such as Bubble Sort, Selection Sort, Merge Sort, Quick Sort, Heap Sort.

- In CS101 course, we only cover basic sorting algorithm which is insertion sort

- Contents covered till now (related to arrays)
  - Declaring arrays, initializing arrays
  - Performing operations on arrays (add, subtract, etc)
  - Passing arrays to functions (elements of array, whole array, and const array)
  - Sorting and Searching on arrays

Pointers

- Pointer variables
  - Contain memory addresses as their values

- Normal variables contain a specific value (direct reference)

- Pointers contain the address of a variable that has a specific value (indirect reference)

Pointers

- Syntax:

```
dataType *identifier;
```

- Examples:

```
int *p;
char *ch;
```

- These statements are equivalent:

```
int  *p;
int*  p;
int * p;
```

Declaring pointer variables

- In the statement:

```
int* p, q;
```

  - Only **p** is a pointer variable
  - **q** is an **int** variable

- To avoid confusion, attach the character ***** to the variable name:

```
int *p, q;
int *p, *q;
```

Address of Operator (&)

- Address of operator (**&**):
  - A unary operator that returns the address of its operand
- Example:

```
int x;
int *p;
p = &x;
```

  - Assigns the address of **x** to **p**

Dereferencing Operator (*)

- Dereferencing operator (or indirection operator):
  - When used as a unary operator, * refers to object to which its operand points

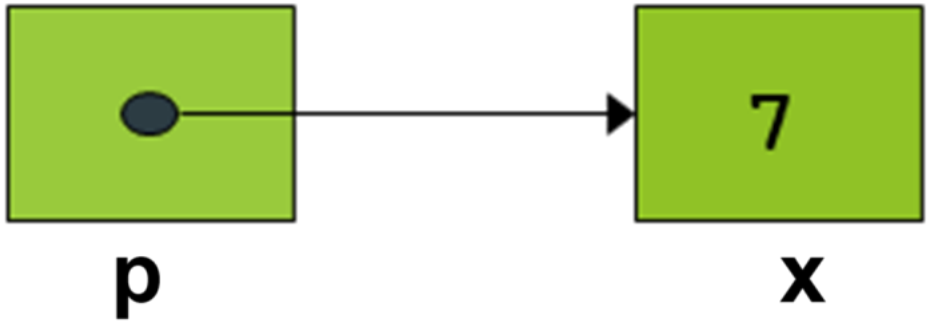
- Example:

```
cout << *p << endl;
```

  - Displays the value stored in the memory location pointed by **p**

Graphical Sketch of Pointer

```
int x;
int *p;
p = &x;
```



Pointers and Addresses

- C++ allows two ways of accessing variables
  - Name (C++ keeps track of the address of the first location allocated to the variable)
  - Address/Pointer
- Symbol **`&`** gets the address of the variable that follows it
- Addresses/Pointers can be displayed by the **`cout`** statement
  - Addresses are displayed in HEXADECIMAL

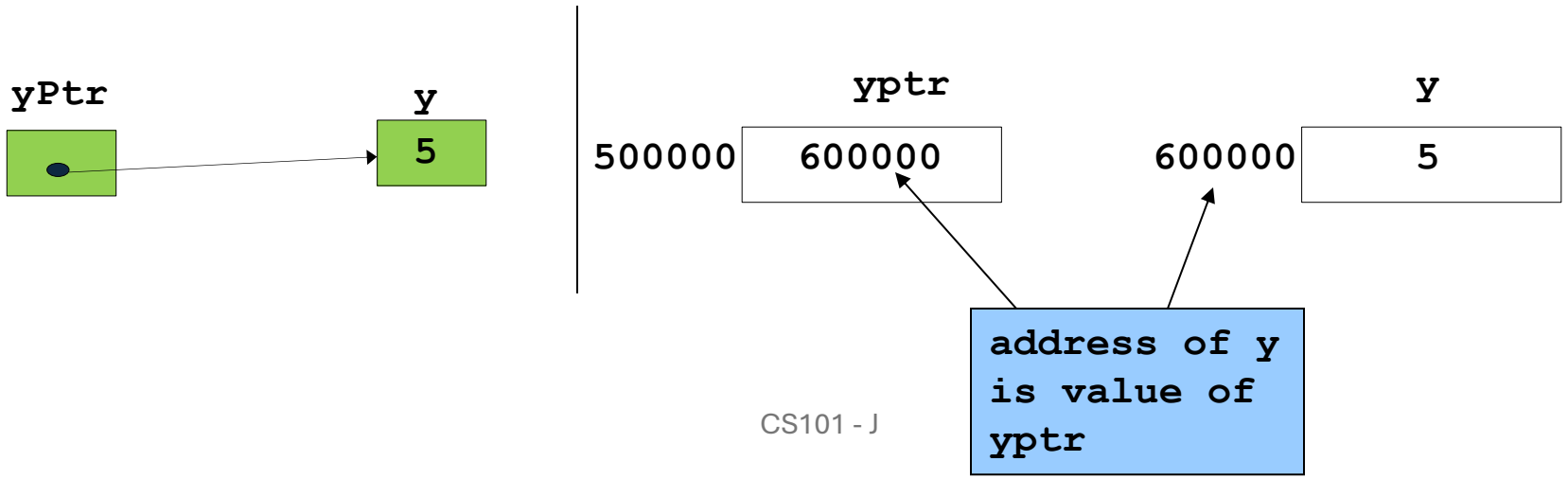
wk11s25.cpp

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

int main()
{
        int y;
        int* yptr;
        yptr=&y;
        y=7;

        cout<<"The value of y is"<<y<<endl;
        cout<<"The value of y_ptr is "<<*yptr<<endl;

        cout<<"The address of y is"<<&y<<endl;
        cout<<"The address of y is "<<yptr;

        return 0;
}
```

Pointer Initialization

- Can declare pointers to any data type
- Pointer initialization
  - Initialized to **0**, **NULL**, or an address. **0** or **NULL** points to nothing
- Example

```
int y = 5;
int *yPtr;
yPtr = &y;     // yPtr gets address of y
```




GHULAM ISHAQ KHAN INSTITUTE OF ENGINEERING SCIENCES AND TECHNOLOGY
PAKISTAN