COMSATS University
Islamabad
(Lahore Campus)

CSC103-Programming Fundamentals

MS. MAHWISH WAQAS

MAWISH.WAQAS@CUILAHORE.EDU.PK

Chapter 8
Arrays and Strings

Base Address of an Array and Array in Computer Memory

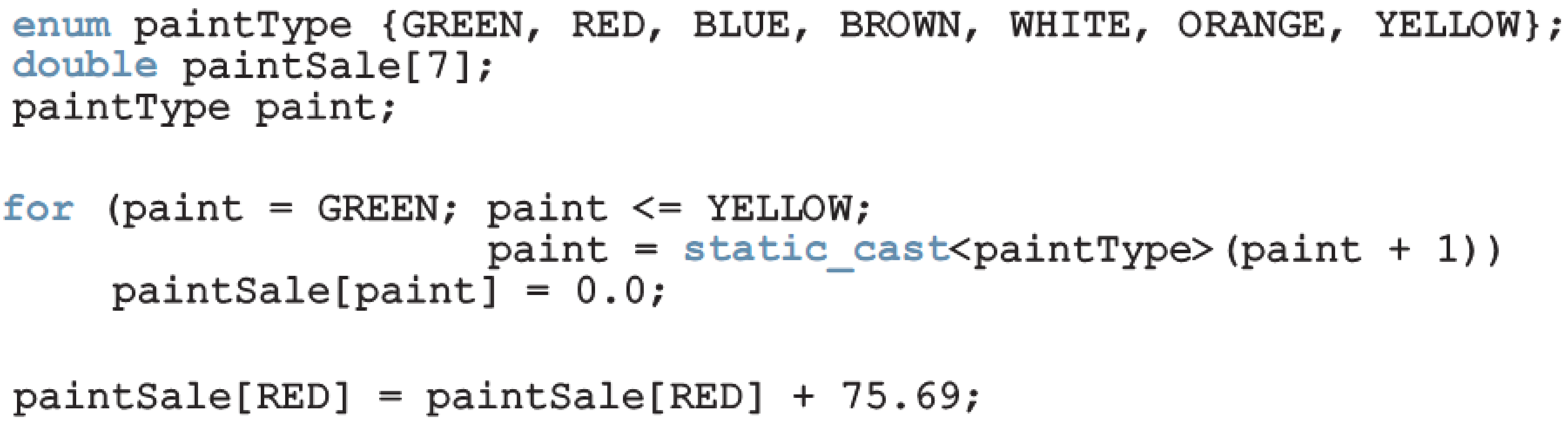
- Base address of an array: address (memory location) of the first array component
- Example:
  - If `list` is a one-dimensional array, its base address is the address of `list[0]`
- When an array is passed as a parameter, the base address of the actual array is passed to the formal parameter

Functions Cannot Return a Value of the Type Array

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

Integral Data Type and Array Indices

- C++ allows any integral type to be used as an array index
  - Improves code readability
- Example:



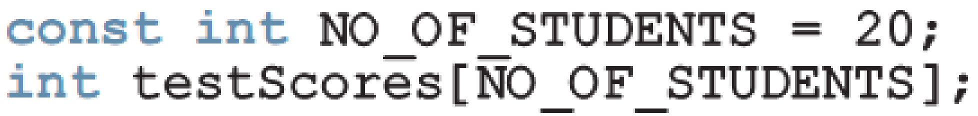
```
enum paintType {GREEN, RED, BLUE, BROWN, WHITE, ORANGE, YELLOW};
double paintSale[7];
paintType paint;

for (paint = GREEN; paint <= YELLOW;
                    paint = static_cast<paintType>(paint + 1))
    paintSale[paint] = 0.0;

paintSale[RED] = paintSale[RED] + 75.69;
```

Other Ways to Declare Arrays

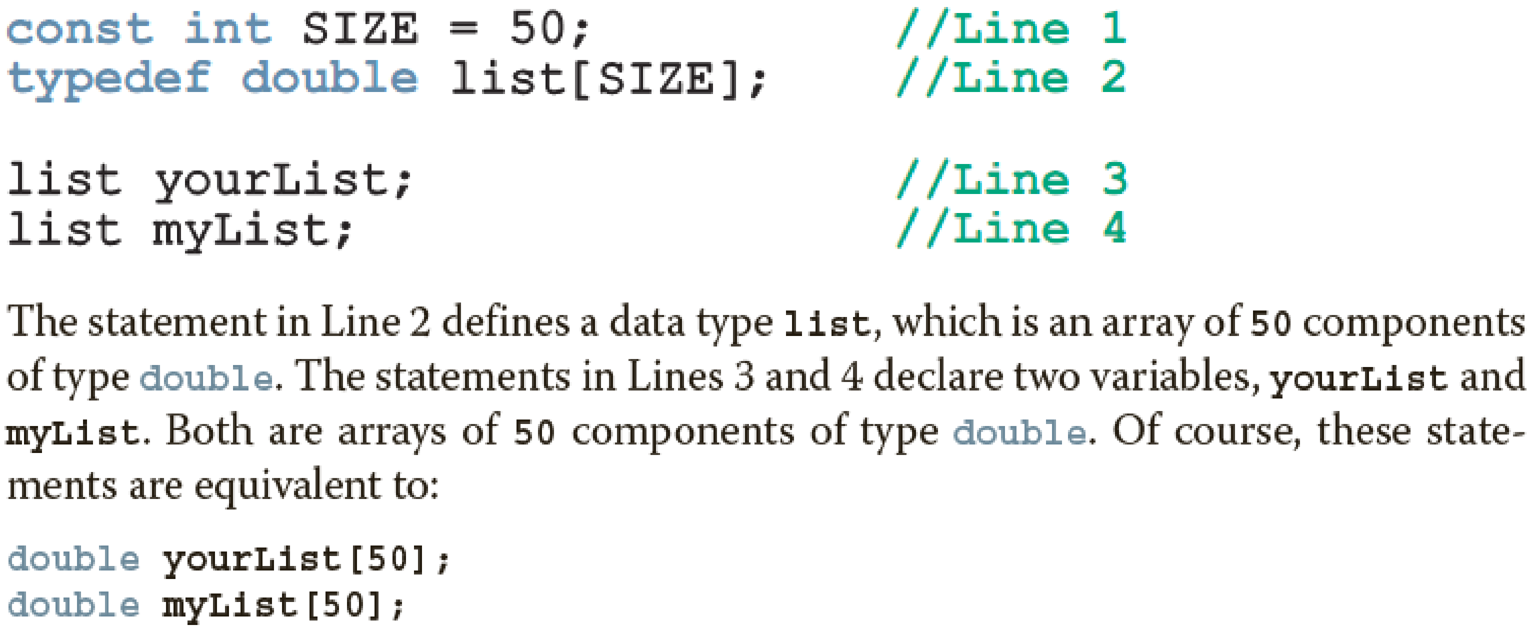
- In C++, you can create synonyms or aliases to a previously defined data type by using the typedef statement.
- Syntax : typedef existingTypeName newTypeName;
- In C++, typedef is a reserved word. The typedef statement does not create any new data type; it creates only an alias to an existing data type.

Other Ways to Declare Arrays

- Examples:



```
const int NO_OF_STUDENTS = 20;
int testScores[NO_OF_STUDENTS];
```



```
const int SIZE = 50;            //Line 1
typedef double list[SIZE];      //Line 2

list yourList;                  //Line 3
list myList;                    //Line 4
```

The statement in Line 2 defines a data type `list`, which is an array of `50` components of type `double`. The statements in Lines 3 and 4 declare two variables, `yourList` and `myList`. Both are arrays of `50` components of type `double`. Of course, these statements are equivalent to:

```
double yourList[50];
double myList[50];
```

Searching an Array for a Specific Item

- 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 list

```
int seqSearch(const int list[], int listLength, int searchItem)
{
 int loc;
 bool found = false;
 loc = 0;
 while (loc < listLength && !found)
         if (list[loc] == searchItem)
                  found = true;
         else
                  loc++;
         if (found)
                  return loc;
         else
                  return -1;
 }
```

Sorting

- Selection sort: rearrange the list by selecting an element and moving it to its proper position
- Steps:
  - Find the smallest element in the unsorted portion of the list
  - Move it to the top of the unsorted portion by swapping with the element currently there
  - Start again with the rest of the list

Selection Sort (cont'd.)

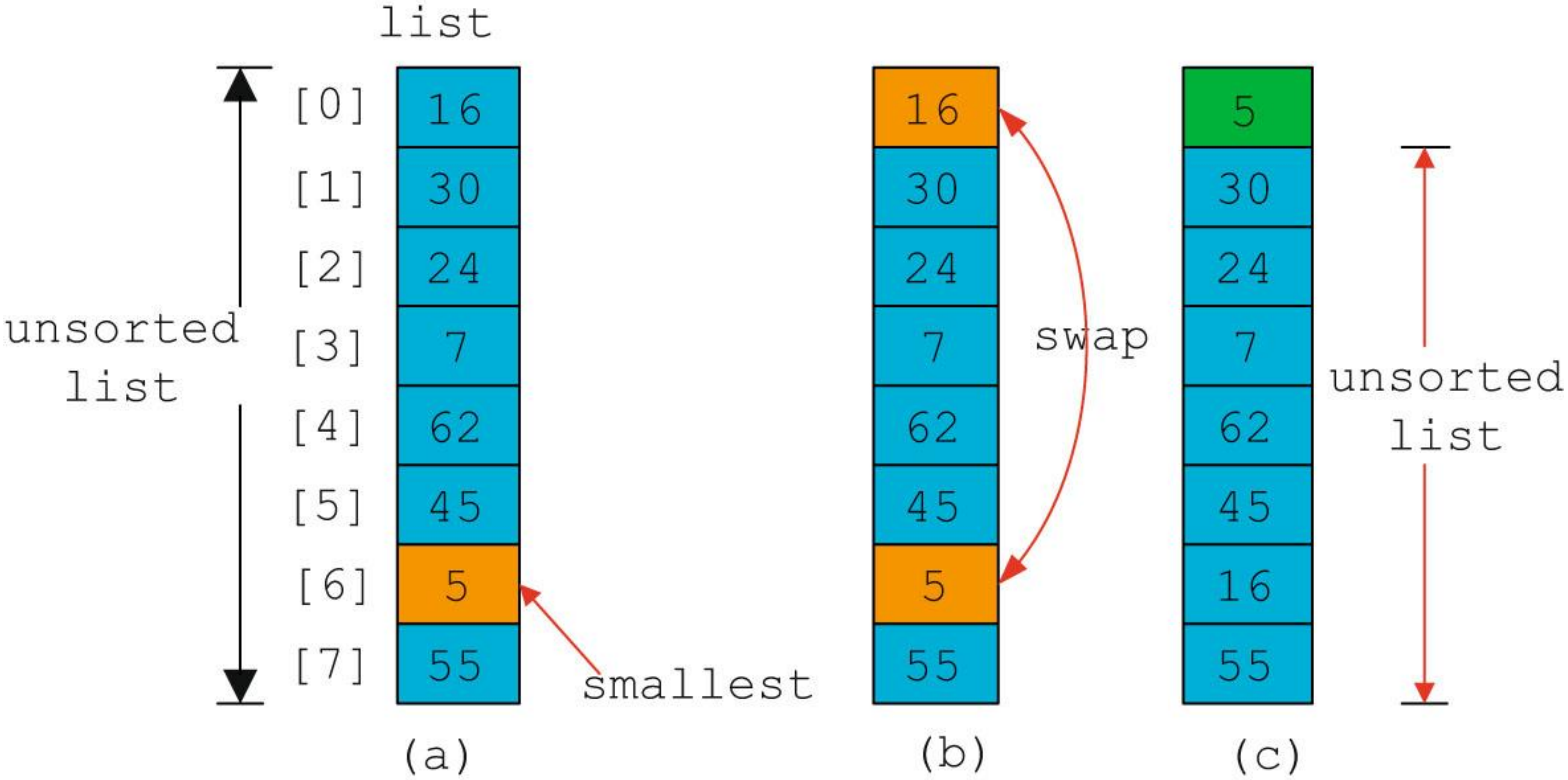
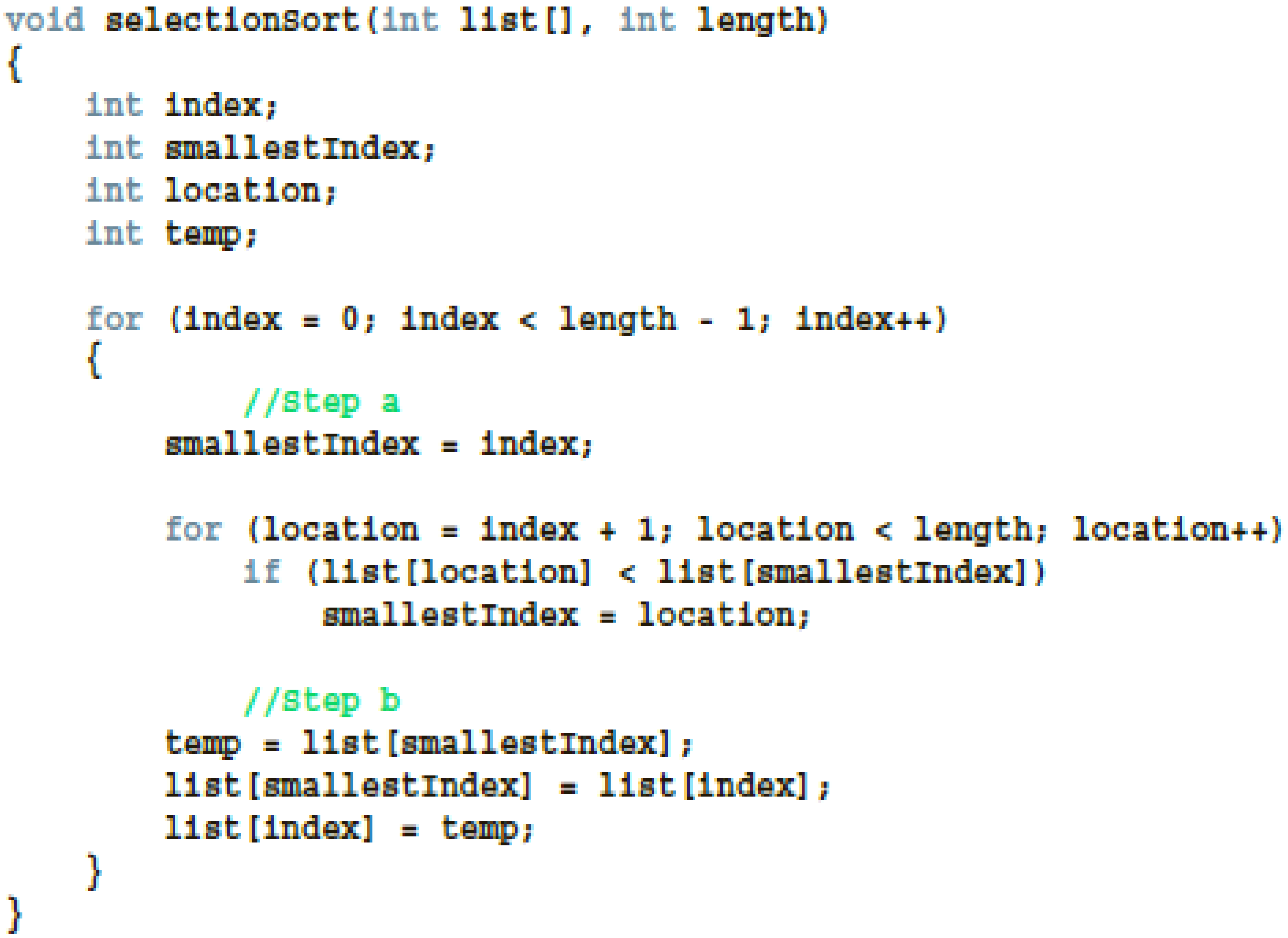


FIGURE 8-10 Elements of `list` during the first iteration



```
void selectionSort(int list[], int length)
{
    int index;
    int smallestIndex;
    int location;
    int temp;

    for (index = 0; index < length - 1; index++)
    {
            //Step a
        smallestIndex = index;

        for (location = index + 1; location < length; location++)
            if (list[location] < list[smallestIndex])
                smallestIndex = location;

            //Step b
        temp = list[smallestIndex];
        list[smallestIndex] = list[index];
        list[index] = temp;
    }
}
```