Arrays

Chapter 4

1D Arrays

Consider the following programs

- Create a program that can store the grades for all assignments and students in a class
- Write a program that stores a sentence as a series of characters
- Create a database customers in which their names, phone numbers, and address are stored
- Store 100 random numbers

- All of these require lots of data to be stored. For instance the last one would require 100 declarations!

Further Examination

- Store 20 grades
  - Requires 20 declarations
  - Every assignment would need to be to a unique variable
  - Entering the assignment
  - As such a 20 case conditional would be required to fill the variables uniquely

```
int i, grade ,g1, g2, g3, g4
……
for(i=0;i<20;i++)
{
        cin >> grade;
        if(i ==0)
           g1 = grade;
        else if(i == 1)
          g2 = grade;
        else if( i == 3)
          g3 = grade;
    …
}
```

Surely there is a better way

Arrays

- Just as there is a need for repetition of code, there is a need for repetition of data
  - i.e. need a way to generically store large amounts of data
- Define: form of data that can hold several values all of the same type
  - i.e. with a single declaration 1000s of variables can be created

Array Declaration

- General Form

```
datatype variable_name[num_of_variables];
```

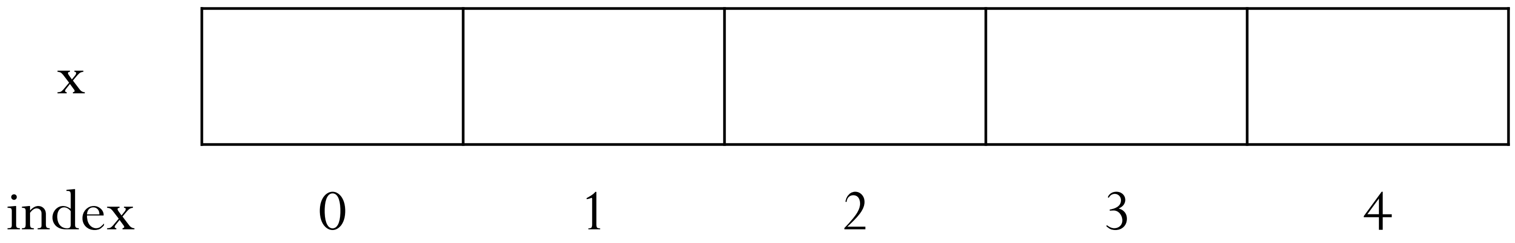
- Example

```
float grades[20];
```

- Creates 20 floats that can be accessed through the variable name grades

Anatomy of an Array

```
int x[5];
```

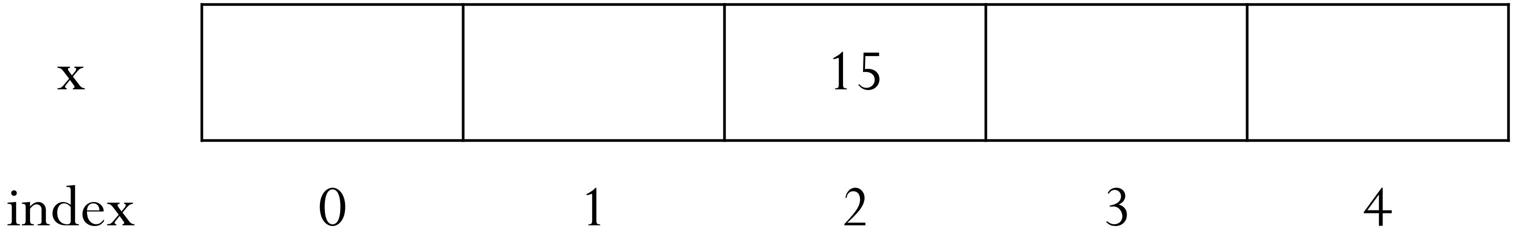


x					
index	0	1	2	3	4

- Since there are many variables contained within x, an index is used to uniquely label each one
- Arrays are zero indexed, meaning the last index is n-1 where n is the size

Store Value in Array

int x[5];

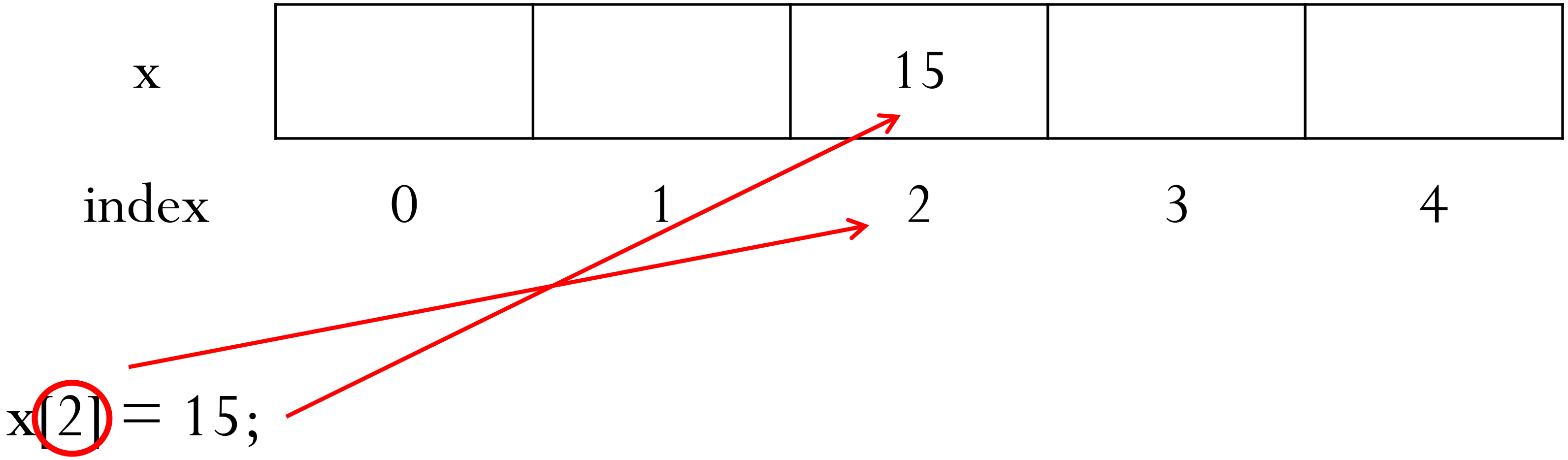


x			15		
index	0	1	2	3	4

x[2] = 15;

Store Value in Array

int x[5];



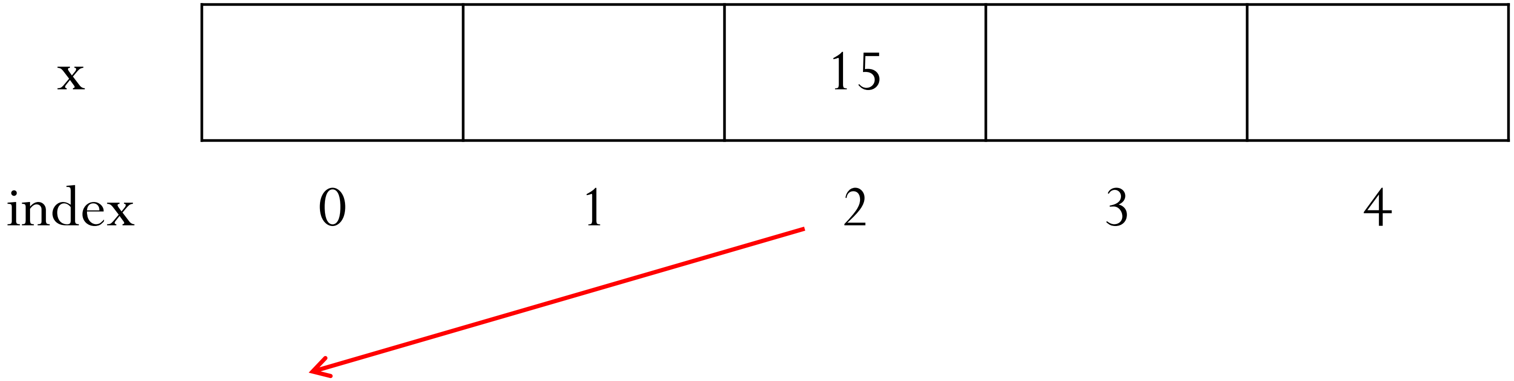
x			15		
index	0	1	2	3	4

x[2] = 15;

- x[2] indicates the location in the array
- As usual the number to the right of the equals sign is the value to be stored

Reading Value in Array

```
int x[5];
```

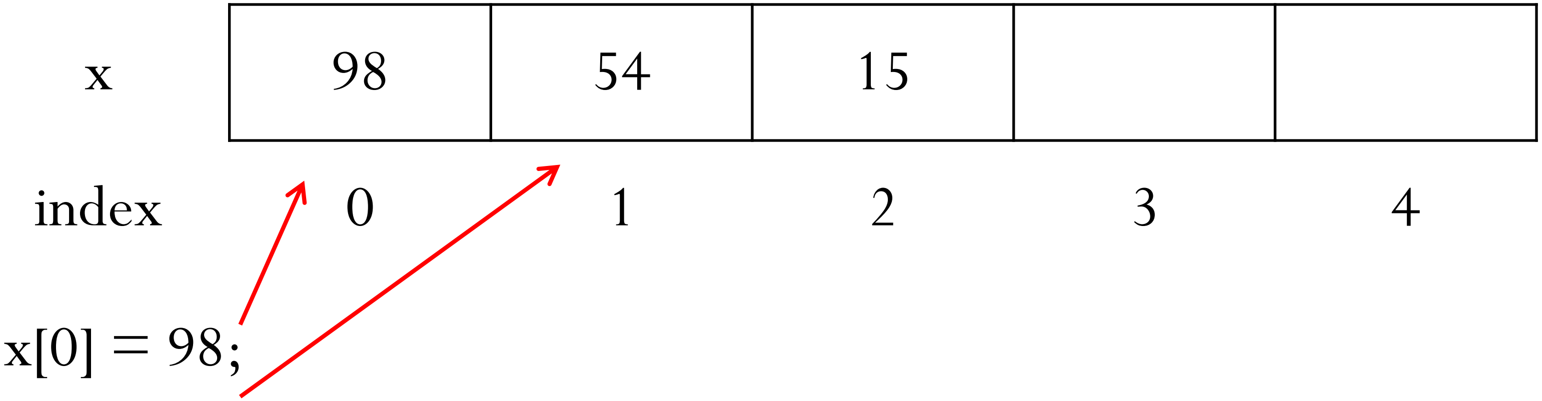


x			15		
index	0	1	2	3	4

```
cout << x[2] << endl;
```

More Examples

int x[5];



x	98	54	15		
index	0	1	2	3	4

```
x[0] = 98;
x[1] = 54;
```

More Examples

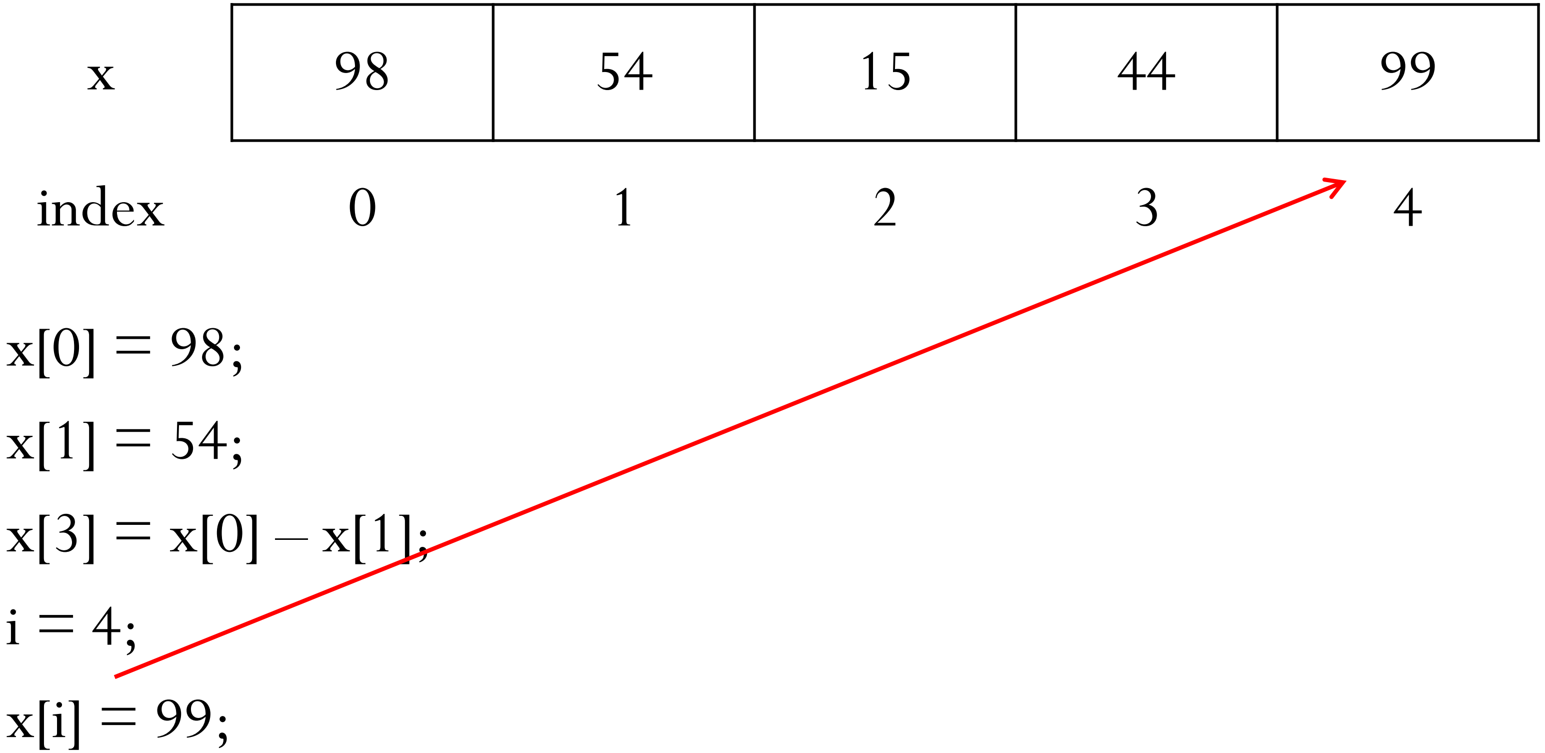
```
int x[5];
```

x	98	54	15	44	
index	0	1	2	3	4

```
x[0] = 98;
x[1] = 54;
x[3] = x[0] – x[1];
```

More Examples

```
int x[5];
```



x	98	54	15	44	99
index	0	1	2	3	4

```
x[0] = 98;
x[1] = 54;
x[3] = x[0] – x[1];
i = 4;
x[i] = 99;
```

Rules of Arrays

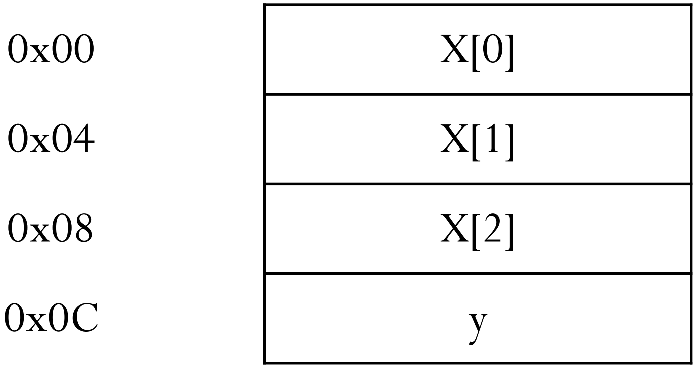
- Arrays must be accessed using the index notation!!!!
  - i.e. variableName[index of element]
  - an array of size n has a range of variableName[0] to variableName[n-1]
- The size of the array must be known at compile time
  - i.e. int x[4]; is legal
  - i.e. the following is illegal

```
int i = 3;
int x[i];
```

Don’ts of Arrays

- Accessing out of bounds (i.e. accessing element n instead of n-1) can cause two types of bugs
  - Crash the program
  - Access another variable in the program because a variable maybe in the next memory position
    - in this case y maybe accessed

```
int x[3];
int y;
```



0x00	X[0]
0x04	X[1]
0x08	X[2]
0x0C	y

Memory

Don’ts of Arrays

- Using an array without index notation. Some classic mistakes are listed below.

```
int x[2], y[2];
x[0] = 10;
x[1] = 15;

cout << x << endl;   Doesn’t print the contents of the array
x = 6;  Doesn’t initialize the contents of the array to 6,
y = x;  Doesn’t make a copy of x
```

Example 6.1

- Write a program to read in and store 5 grades. Then take the average of the 5 grades and print it.

Example 6.1 Solution

```
float grades[5], sum;
int i;

// Read in grades
for(i = 0; i < 5; i++)
{
   cout << "Enter a grade: ";
   cin >> grades[i];
}

// To Sum Grades
for(i = 0, sum = 0; i < 5; i++)
   sum += grades[i];

cout << "Average is: " << sum / 5 << endl;
```

Unguided Example 6.2

- Write a program to fill a 10 element array of integers with random numbers that fall between 1 and 100. Then print the contents of the array using a loop.

Note:

use srand(time(NULL)); once at the beginning of the program to seed the random number generator. Otherwise you will generate the same set of random numbers each time.

rand() generates a random number from 0 to something huge

#include<cstdlib>

#include<ctime>

Functions and Arrays

- Below is an example of passing a 1D Array to a function

```
void printArray(int[], int);
int main(){
   int x[10], i;
   for(i = 0; i < 10; i++)
      x[i] = 0;
   printArray(x, 10);
   return 0;
}

void printArray(int arr[], int size)
{
    int i;
    for(i = 0; i < size; i++)
        cout << setw(2) << arr[i];
    cout << endl;
}
```

Passing Arrays

```
void printArray(int arr[], int size)
{
    int i;
    for(i = 0; i < size; i++)
        cout << setw(2) << arr[i];
    cout << endl;
}
```

- Empty brackets indicated the parameter is a 1D array
- Array's dimension does not need to be specified
- Since there is no way to determine an array's size, the size must be passed to the function

Arrays and Pass by Value

```
void incArray(int[], int);
int main(){
   int x[10], i;
   for(i = 0; i < 10; i++)
      x[i] = 0;
   incArray(x, 10);
   return 0;
}

void incArray(int arr[], int size)
{
   int i;
   for(i = 0; i < 10; i++)
      arr[i]++;
}
```

- Will the contents of array x change after calling incArray?

Yes

Arrays and Pass by Value

- Since you are passing a memory pointer arrays are being passed by reference
- Therefore modifications made to arrays inside of a function will be reflected back into the original array
- We will investigate this further when pointers are introduced

Returning Arrays

```
int[] initArray();
int main(){
   int x[10], i;
   x = incArray();
   return 0;
}

int[] initArray()
{
   int arr[10], i;
   for(i = 0; i < 10; i++)
       arr[i] = 0;
   return arr;
}
```

- To the right is a function that initializes and returns an array
- Does the code work?
  - **NO**
- Since arr is a memory pointer the contents will not be returned
- Furthermore, the pointer will be to a array that no longer exists after exiting the function
- Rule: **Static arrays** cannot be returned using a return statement
- Practice is to pass them as an input or to use dynamic arrays

End Note

- It should be noted that the repetition of data is usually accompanied by repetition of code.

- Almost all algorithms working on arrays will require a loop