Arrays

Course Code: CSC1102 &1103 Course Title: Introduction to Programming

Dept. of Computer Science
Faculty of Science and Technology

Lecturer No:	**5**	**Week No:**	**3 (1X1.5 hrs),** **4 (2X1.5 hrs)**	**Semester:**	
Lecturer:	*Name & email*				

Lecture 5: Outline

- Arrays
- The concept of array
  - Defining arrays
  - Initializing arrays
  - Character arrays
  - Variable length arrays

The concept of array

- ❑ Array: a set of ordered data items
- ❑ You can define a variable called x, which represents not a *single* value, but an entire *set of values*.
- ❑ Each element of the set can then be referenced by means of a number called an *index* number or *subscript*.
- ❑ Mathematics: a subscripted variable, x_i, refers to the *i*th element *x* in a set
- ❑ Programming: the equivalent notation is x[i]

Declaring an array

- ❑ Declaring an array variable:
  - ❑ Declaring the **type of elements** that will be contained in the array—such as int, float, char, etc.
  - ❑ Declaring the **maximum number of elements** that will be stored inside the array.
    - ❑ The compiler needs this information to determine how much memory space to reserve for the array.)
    - ❑ This must be a **constant integer value**

- ❑ The range for valid index values:
  - ❑ First element is at index 0
  - ❑ Last element is at index [size-1]
  - ❑ It is the task of the programmer to make sure that array elements are referred by indexes that are in the valid range ! The compiler cannot verify this, and it comes to severe runtime errors !

Arrays - Example

```
int values[10];
```

Declares an array of 10 elements of type `int`

Using Symbolic Constants for array size:

```
#define N  10
…
int values[N];
```

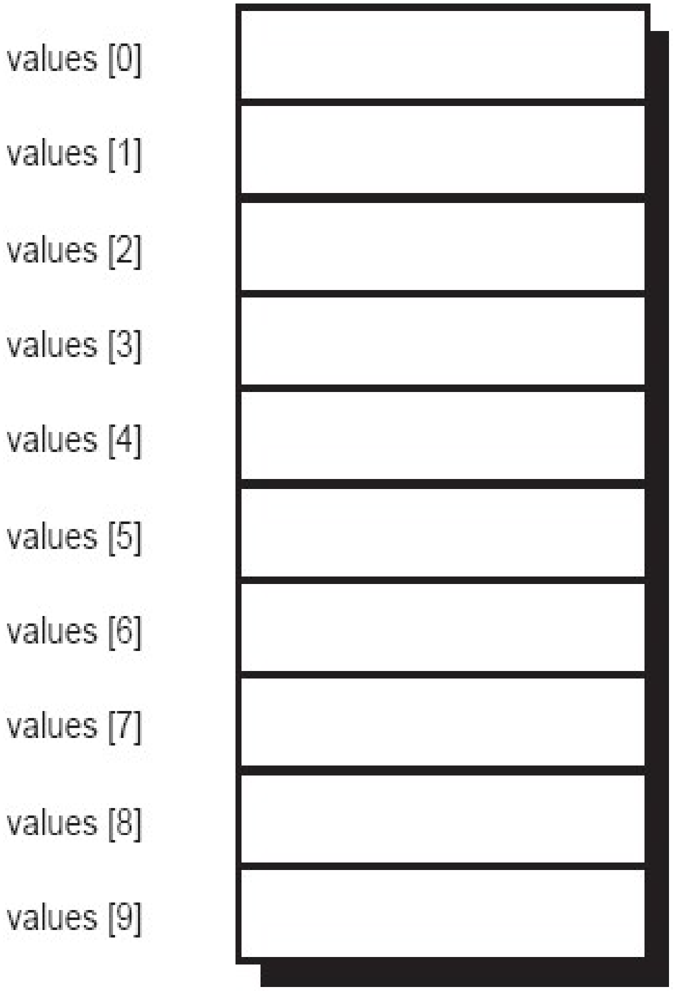
Valid indexes:

```
values[0]=5;
values[9]=7;
```

Invalid indexes:

```
values[10]=3;
values[-1]=6;
```

In memory: elements of an array are stored at consecutive locations



Arrays - Example

```
#include <iostream>
#define N  6
int main (void)
{
    int values[N];
    int index;
    for ( index = 0; index < N; ++index ) {
        cout<<“Enter value of element”<<index<<endl;
        cin>>values[index];
        }
    for ( index = 0; index < N; ++index )
       cout<< “values[“<<i<<“]=“<< values[index]<<endl;
    return 0;
}
```

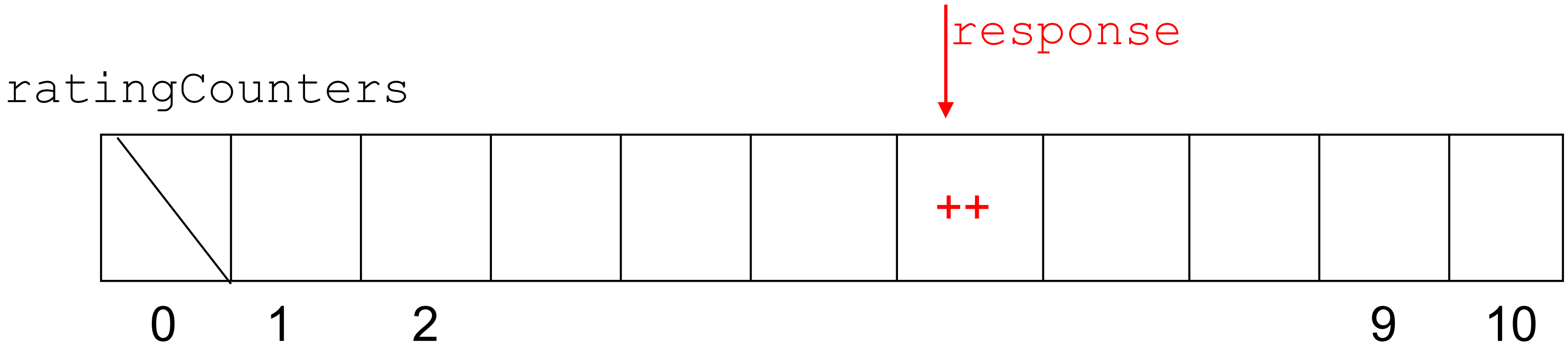
Using symbolic constants for array size makes program more general

Typical loop for processing all elements of an array

What goes wrong if an index goes out of range ?

```
#include <iostream>
using namespace std;
int main (void){
int NA, NB;
cout<<"Enter NA and NB"<<endl;
cin>>NA>>NB;
int b[NB],a[NA];
int index;
for ( index = 0; index < NB; index++ )
    b[index]=10+index;
for ( index = 0; index < NA+2; ++index )
    a[index]=index;
for ( index = 0; index < NA+2; ++index )
       cout<<"a ["<<index<<"]=  "<<a[index]<<endl;
for ( index = 0; index < NB; ++index )
    cout<<"b ["<<index<<"] ="<< b[index]<<endl;
return 0;
}
```

Exercise: Array of counters



ratingCounters[i] = how many persons rated the show an i

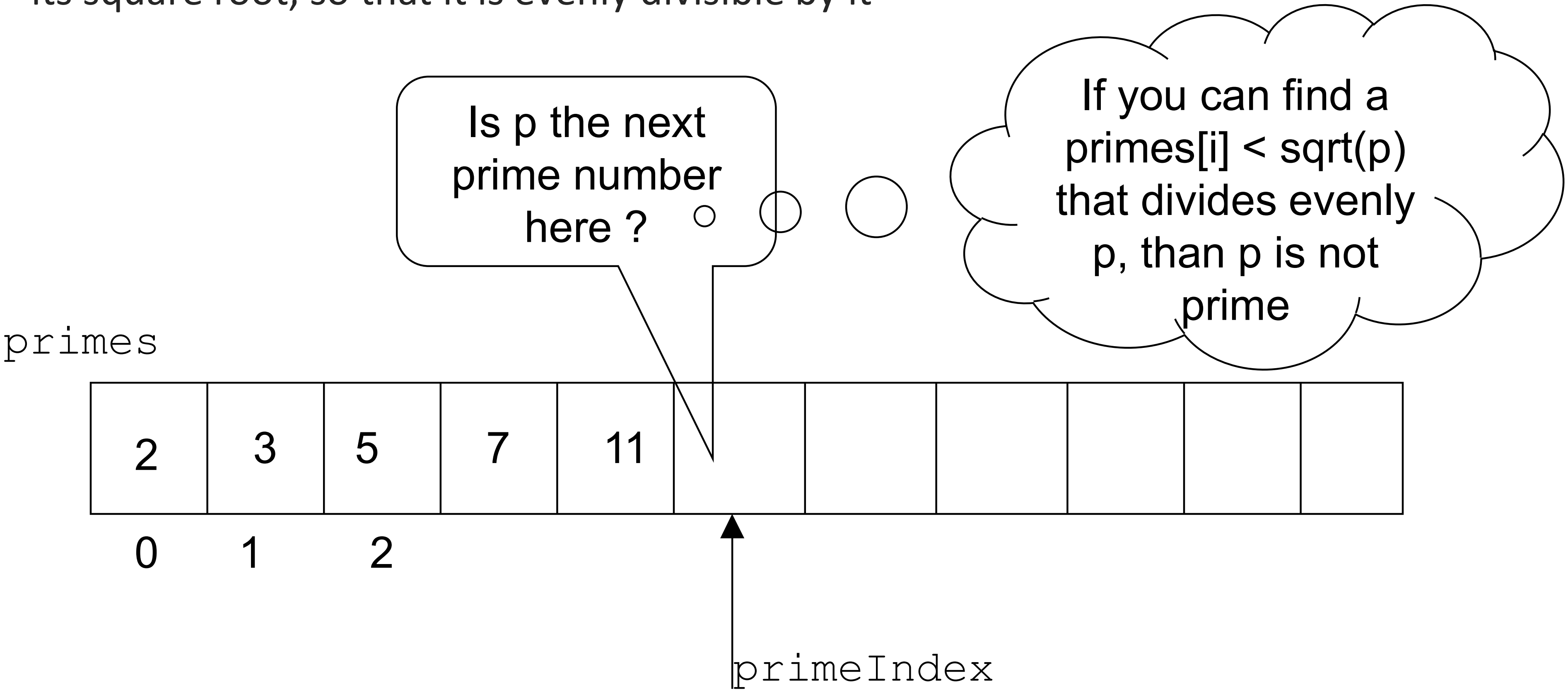
Exercise: Fibonacci numbers

```
// Program to generate the first 15 Fibonacci numbers
#include <iostream>
using namespace std;
int main (void)
{
int Fibonacci[15], i;
Fibonacci[0] = 0; // by definition
Fibonacci[1] = 1; // ditto
for ( i = 2; i < 15; ++i )
      Fibonacci[i] = Fibonacci[i-2] + Fibonacci[i-1];
for ( i = 0; i < 15; ++i )
      cout<< "Fibonacci["<<i<<"]="<<Fibonacci[i]<<endl;
return 0;
}
```

Exercise: Prime numbers

- An improved method for generating prime numbers involves the notion that a number p is prime if it is not evenly divisible by any other prime number

- Another improvement: a number p is prime if there is no prime number smaller than its square root, so that it is evenly divisible by it



Is p the next prime number here ?

If you can find a primes[i] < sqrt(p) that divides evenly p, than p is not prime

`primes`

2	3	5	7	11						
0	1	2								

`primeIndex`

Exercise: Prime numbers

```
#include <iostream>
using namespace std;
// Modified program to generate prime numbers
int main (void)  {
int p, i, primes[50], primeIndex = 2; bool isPrime;
primes[0] = 2; primes[1] = 3;
for ( p = 5; p <= 50; p = p + 2 ) {
       isPrime = true;
for ( i = 1; isPrime && p / primes[i] >= primes[i]; ++i )
             if ( p % primes[i] == 0 )
                    isPrime = false;
       if ( isPrime == true ) {
             primes[primeIndex] = p;
             ++primeIndex;
             }
}
for ( i = 0; i < primeIndex; ++i )
       cout<<"Primes["<<i<<"]"<<primes[i]<<endl;
return 0;
}
```

Initializing arrays

- int counters[5] = { 0, 0, 0, 0, 0 };

- char letters[5] = { 'a', 'b', 'c', 'd', 'e' };

- float sample_data[500] = { 100.0, 300.0, 500.5 };

- The C++ language allows you to define an array without specifying the number of elements. If this is done, the size of the array is determined automatically based on the number of initialization elements: int counters[] = { 0, 0, 0, 0, 0 };

Character arrays

```
#include <stdio.h>
int main (void)
{
    char word[] = { 'H', 'e', 'l', 'l', 'o', '!' };
    int i;
    for ( i = 0; i < 6; ++i )
        cout<<word[i]);
    return 0;
}
```

a special case of character arrays: the character *string* type =>in a later chapter

Example: Base conversion using arrays

```
#include <iostream>
using namespace std;
int main (void)
{
const char baseDigits[16] = {
'0', '1', '2', '3', '4', '5', '6', '7',
'8', '9', 'A', 'B', 'C', 'D', 'E', 'F' };
int convertedNumber[64];
long int numberToConvert;
int nextDigit, base, index = 0;
// get the number and the base
cout<<"Number to be converted? "<<endl;
cin>>numberToConvert;
cout<<"Base? "<<endl;
cin>>base;
```

Example continued

```
// convert to the indicated base
do {
      convertedNumber[index] = numberToConvert % base;
      ++index;
      numberToConvert = numberToConvert / base;
}
while ( numberToConvert != 0 );
// display the results in reverse order
cout<<"Converted number = ";
for (--index; index >= 0; --index )
    {
      nextDigit = convertedNumber[index];
      cout<<baseDigits[nextDigit];
      }
return 0;
}
```

new and delete operators in C++ for dynamic memory

- ❑ Dynamic memory allocation in C++ means allocating memory manually by programmer when needed.
- ❑ **How is it different from memory allocated to normal variables?**

  Previously used variables like “int a”, “char str[10]”, etc., memory is automatically allocated and deallocated. For dynamically allocated memory it is programmers responsibility to deallocate memory when no longer needed. If programmer doesn’t deallocate memory, it causes memory leak (memory is not deallocated until program terminates).
- ❑ **How is memory allocated/deallocated in C++?**

  C++ has two operators new and delete that perform the task of allocating and freeing the memory. The allocation is done at runtime.

Example: Variable length arrays

```
#include <iostream>
using namespace std;
int main() {
    int n;
    int i;
    cout<<"How many elements do you have? "<<endl;
    cin>>n;

    int *a = new int[n];

    for(i = 0; i < n; i++)
           cin>>a[i];
    for(i = 0; i < n; i++)
           cout<< a[i];

    delete a;
    return 0;
}
```

Pointer is used so that we know where the memory is allocated.

new operator is responsible for allocating the memory

It indicates, memory is allocated for this type of datatype, here it is int

If we want to allocate memory for multiple data’s we can create array by providing any positive integer number or a variable having similar type value

Now when we are done using our allocated memory which is where a pointer is pointing ,we must deallocate it using delete operator.