Arrays and Loops

Array – A “fundamental data structure” for you to better organize your data

Outline

- Arrays
  - **Basic concepts**
  - Two basic operations:
    - How to create an array
    - How to use (access) an array
  - Precaution and Common mistakes
- MISC topics and multi-dimensional arrays
- Example Algorithms using Loops and Arrays

Why array?

Ordinary Variable

Like a box for storing one single value

```
int score ;
```

How if you need to store the scores of 50 students in the class?

Shall we create **50 variables**?

```
// 50 variables
int score1 ;
int score2 ;
int score3 ;
int score4 ;
int score5 ;
…
int score49 ;
int score50 ;
```

How to process them?

e.g., compute the average

How if we add one more student to the class?

Need to change code?

```
sum  = score1 + score2 + ... ;
aver = sum / 50.0 ;
```

Oh No!!!

(1) What is an array?

- A collection of ordered data
  - In C: homogeneous, i.e., same type of data

Ordinary Variable

Like a box for storing one single value

```
int score ;
```

```
int scores[100] ;
int num_students ;
```

Array

Like a cabinet with many drawers. Each drawer stores only a single value.

It is an **ordered** list of data values; by **indexing** them, we can refer to values in

- 1st drawer,
- 2nd drawer,
- 3rd drawer, etc.

Note: some programming languages support data structures with heterogeneous data, e.g., list in Python

What is an array?

- An array is a **data type** that stores a finite number of "***variables***" of the same data type.

Point 1. The length of an array is set when it is created (**static** allocation).
Important. The length is fixed (i.e., cannot change) throughout its lifetime.

Point 2. The length of an array determines the ***number of elements*** inside.
Important. All elements must be of the same type.

Index	0	1	2	3	4	5	6	7
Elements	5	6	8	5	6	8	7	1

Let us assume that the above is an array of "**`int`**" type of 8 elements.

Outline

- Arrays
  - Basic concepts
  - **Two basic operations (like variables):**
    - **How to create an array**
    - **How to use (access) an array**
  - Precaution and Common mistakes
- MISC topics and multi-dimensional arrays
- Example Algorithms using Loops and Arrays

Two Basic Operations: Like Variables

1. Create an array
   - Declaration:

```
int data1    ;   // conventional variable
int data2[5] ;   // an array of variables
```

   - Optionally with initialization

2. Access (read & write) an array
   - Read the value of an array element
   - Modify (write) the value of an array element

#1. Declaration and Initialization

- **Declaration**:

```
int score_array[ 5 ] ;
```

  - In C, once you've created an array, its size is fixed!
  - The size should be **known** during the compilation!

- **Initialization**: say we want to initialize an array, so that it carries 5 integers of the same value = 0

```
int score_array[ 5 ] = { 0 , 0 , 0 , 0 , 0 };
```

Declaration and Initialization

- Initialization Examples
  - Initialize an array without specifying its size:

```
int score_array[] = { 0 , 0 , 0 , 0 };  // array of size 4
```

  - Initialize only the first two array elements with 1:

```
int score_array[ 5 ] = { 1 , 1 };  // others are 0
```

  - Invalid syntax:

```
int score_array[] ; // Error! Need a size or an initializer
```

```
int score_array[ 5 ] = { , , 1 , , }; // Error! Can't skip!
```

```
int score_array[ 5 ] = { 1 , 2 , 3 , 4 , 5 , 6 };
```

#2. Accessing Array Element

- Assign values to array elements:
  - Every element has an index: from **0** to "**length - 1**"
  - Use "`score[i]`" as the **(i+1)-th** element

```
int score[ 5 ];
score[ 0 ] = 0 ;     // first element
score[ 1 ] = 1 ;     // second element
```

- Reading values from an array

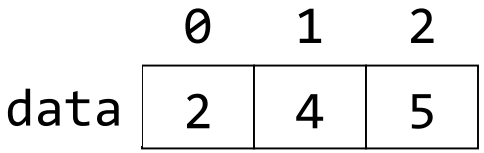
```
printf( "%d\n" , score[ 0 ] );  // print the 1st element.
```

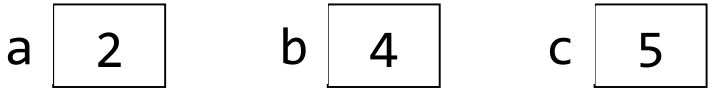
Note: in C, array index starts from "**zero**", but in some languages (e.g., MATLAB), index starts from one

A simple example

```
1  int data[3] ;
2
3  data[0] = 2 ;
4  data[1] = data [0] + 2 ;
5
6  scanf( "%d" , & data[2] ) ;
7
8  // Assume input is 5
9  printf( "%d" , data[2] );
```

```
1  int a , b , c ;
2
3  a = 2 ;
4  b = a + 2 ;
5
6  scanf( "%d" , & c ) ;
7
8  // Assume input is 5
9  printf( "%d" , c );
```





This example illustrates the **“similarity” between an array and ordinary variables**

A simple example

```
1  int data[3] ;
2
3  data[0] = 2 ;
4  data[1] = data [0] + 2 ;
5
6  scanf( "%d" , & data[2] ) ;
7
8  // Assume input is 5
9  printf( "%d" , data[2] );
```

	0	1	2
data	2	4	5

Just like a variable, we need to declare an array before using it.

This statement declares an array to store values of type `int`.

The name of the array is **`data`**.

The array *size* is 3.

A simple example

```
1  int data[3] ;
2
3  data[0] = 2 ;
4  data[1] = data [0] + 2 ;
5
6  scanf( "%d" , & data[2] ) ;
7
8  // Assume input is 5
9  printf( "%d" , data[2] );
```

A single array element is like an ordinary variable.

`data[0]` refers to the 1st element in array **`data`**.

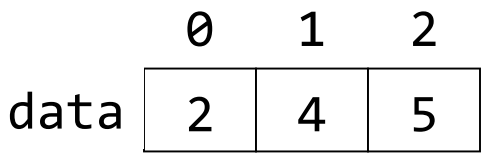
	0	1	2
data	2	4	5

A simple example

```
1  int data[3] ;
2
3  data[0] = 2 ;
4  data[1] = data [0] + 2 ;
5
6  scanf( "%d" , & data[2] ) ;
7
8  // Assume input is 5
9  printf( "%d" , data[2] );
```

In an array declaration, the number in […] indicates the size of an array.

In an expression, the number in […] indicates the index of an array element.



Accessing Array Element

- **Array initialization**
  - You MUST give every element an initial value because every variable starts with unknown (or random) values.

```
  1 int main( void )
  2 {
  3     int score[ 100 ];
  4     int i ;
  5     score[ 0  ] = 0  ;   // assign a value to the first element
...     ......
104     score[ 99 ] = 99 ;   // assign a value to the last element
105
106     for( i = 0 ; i < 100 ; i++ )
107         score[ i ] = i ;  // same as the above code.
108
109     return 0 ;
110 }
```

Accessing Array Element

- **Big picture**
  - It is very typical to use **for-loops** to read and to assign elements in an array.

```
 1  int main( void )
 2  {
 3      int score[ 100 ];
 4      int i ;
 5
 6      for( i = 0 ; i < 100 ; i++ )
 7          score[ i ] = i ;  // (i+1)-th element store the value i
 8
 9      for( i = 0 ; i < 100 ; i++ )
10          printf( "%d\n" , score[ i ] );  // read element value
11
12      return 0 ;
13  }
```

Accessing Array Element

- Index can be an expression of **integral type**
  - E.g., score[i+1] and score[i*2+10]
  - But…… (see next page)

```
 1  int main( void )
 2  {
 3      int score[ 100 ];
 4      int i ;
 5
 6      for( i = 0 ; i < 100 ; i++ )
 7          score[ i ] = i ;  // (i+1)-th element store the value i
 8
 9      for( i = -1 ; i < 99 ; i++ )  // just demo, not good practice!
10          printf( "%d\n" , score[ i+1 ] );  // read element value
11
12      return 0 ;
13  }
```

Outline

- Arrays
  - Basic concepts
  - Two basic operations:
    - How to create an array
    - How to use (access) an array
  - **Precaution and Common mistakes**
- MISC topics and multi-dimensional arrays
- Example Algorithms using Loops and Arrays

Demonstration

- What if **we didn't initialize an array** and print out its content?

- What if we access an array **with incorrect indices**?
  - E.g.,

```
int score[ 5 ];
score[ -1 ] = 0 ;    // an index too small?
score[  5 ] = 0 ;    // an index too big?
```

Note: the consequence of the error is ***unpredictable***.

The program may crash with "*array index out of bounds*"

Other variables may get modified unknowingly.

Recall: program errors

1. **Compilation error**
   - Occurs when you compile a program
   - Compiler can't translate your code to machine code
2. **Runtime error**
   - Occurs only when you run a program
   - There are different types of runtime error:
     - **Segmentation fault and Exception**
       - Because the fault is **too severe** that the program should stop
         E.g., division by zero & "**array out of bound**"
3. **Logic or semantic error**
   - The program shows **wrong behavior**, but it can run!!!
     E.g., circle_area = radius * radius ; // missing pi

Note: a program that can run doesn't mean that it is correct!

Common Exception Scenario

- Typical array out-of-bound exception.
  - Can you find the error?

```
1  int main( void )
2  {
3      int array[ 10 ] = { 1 , 2 , 3 , 4 , 5 , 6 , 7 , 8 , 9 , 10 };
4      int i ;
5
6      for( i = 0 ; i <= 10 ; i++ )
7          array[ i ] = i ;
8
9      return 0 ;
10 }
```

Remember? One more or one less iteration can kill your program!!!

Another Common Issue!

- When creating an array, its size MUST be known!!!
  - Can you find the error?

```
1  int main( void )
2  {
3      int N ;
4      scanf( "%d" , & N );
5      int array[ N ] ;          // DON'T do this!!!
6                                // Not allowed by SOME compilers
7      return 0 ;
8  }
```

Usually, we may initialize an array with max. possible size:

```
const int MAX_SIZE = 20 ;    // let's say there are
int inputs[ MAX_SIZE ] ;     // at most 20 inputs
```

Another Common Issue!

- When creating an array, its size MUST be known!!!
  - After fixing the code:

```
1  const int MAX_SIZE = 20 ;
2
3  int main( void )
4  {
5      int N , array[ MAX_SIZE ] ;  // Declare all variables on top
6                                   // More space for readibility
7      scanf( "%d" , & N );
8
9      return 0 ;
10 }
```

Note: this is **static memory allocation**: **array size MUST be fixed!!!**
In future, we will teach you **dynamic memory allocation** for array

Summary

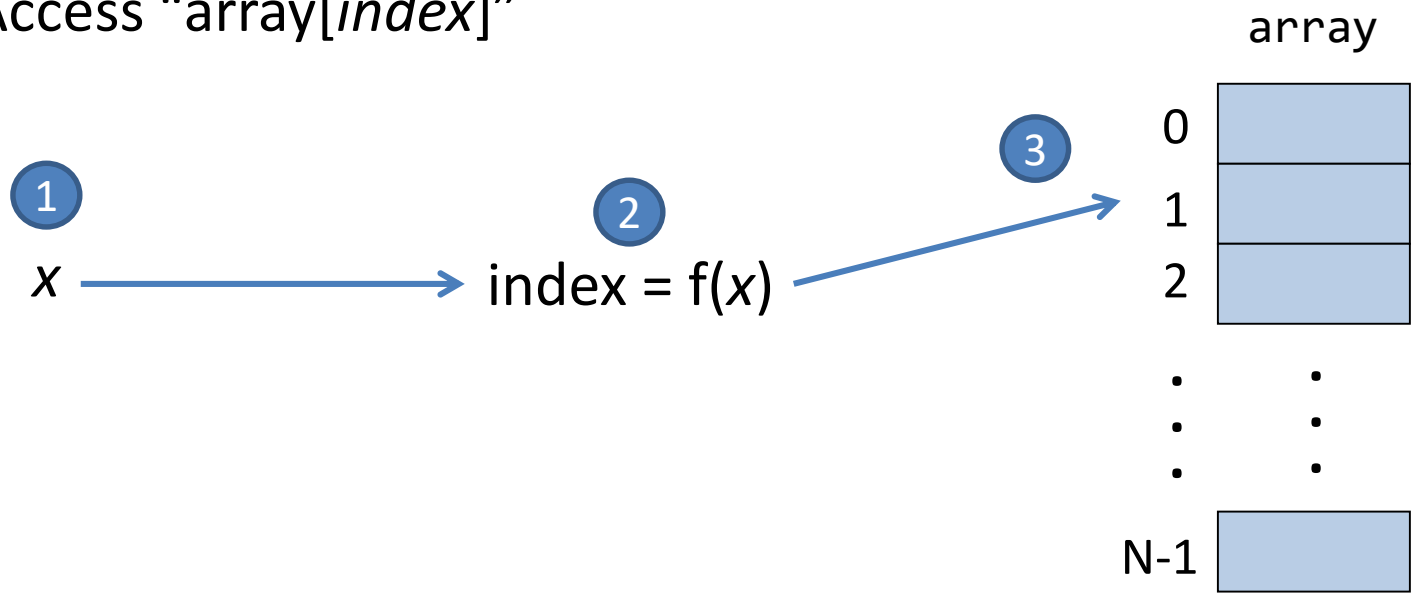
- An array has a **fixed size (number of elements)**.
- An array stores all elements of the **same type**.
  - homogeneous vs. heterogeneous
- An array must be **initialized** before usage.
- The **size** of an array must be **remembered** (has a fixed value) inside the program.

```
const int MAX_SIZE = 20 ;   // let's say there are
int inputs[ MAX_SIZE ] ;    // at most 20 inputs
```

Summary: Array as a Lookup Table

General Idea: Map a value to an array index.

1. Given the "array" as the lookup table, and *x*
2. Calculate the *index* (must be [0, array size-1]) to the array element based on the value of *x*
3. Access "array[*index*]"



Example: for lookup

- Advantage: No need for several **`if-else`** statements

```
1  // monthToDays[month-1] => # of days in "month"
2  int monthToDays[ 12 ] = { 31 , 28 , 31 , 30 , 31 , 30 ,
3                            31 , 31 , 30 , 31 , 30 , 31 } ;
4
5  int month ;         // To store user input
6  int days  ;         // To store # of days in the given month
7
8  scanf( "%d" , & month ) ;
9
10 // assume we forget the existence of leap year for now
11 days = monthToDays[ month - 1 ] ;
   printf( "The input month has %d days!\n" , days ) ;
```

Note: The condition for checking if a year is a leap year is a bit long and is omitted in this example.

Example: for counting

Let's count the number of star ratings of a movie!

```
1   int starCount[ 5 ] = { 0 } ; // all zeros
2   int rating ;                 // To store user input, assuming valid
3
4   // read 4 numbers from the viewers, one-by-one, and tally
5   printf( "Enter 4 viewers' ratings (1 to 5 stars): " );
6   scanf( "%d" , & rating ) ;  starCount[ rating - 1 ]++ ;
7   scanf( "%d" , & rating ) ;  starCount[ rating - 1 ]++ ;
8   scanf( "%d" , & rating ) ;  starCount[ rating - 1 ]++ ;
9   scanf( "%d" , & rating ) ;  starCount[ rating - 1 ]++ ;
10
11  printf( "No. of viewers giving 1 to 5 stars: %d %d %d %d %d\n" ,
12    starCount[ 0 ] ,
13    starCount[ 1 ] ,
14    starCount[ 2 ] ,
15    starCount[ 3 ] ,
16    starCount[ 4 ] ) ;
17
18
```

Can we use “for loops” here?

Example: for counting

Let's count the number of star ratings of a movie!

```
1   int starCount[ 5 ] = { 0 } ; // all zeros
2   int rating , i ;             // To store user input, assuming valid
3
4   // read 4 numbers from the viewers, one-by-one, and tally
5   printf( "Enter 4 viewers' ratings (1 to 5 stars): " );
6   for ( i = 0 ; i < 4 ; i ++ )
7   {
8       scanf( "%d" , & rating ) ;
9       starCount[ rating - 1 ]++ ;
10  }
11
12  printf( "No. of viewers giving 1 to 5 stars:\n" );
13  for ( i = 0 ; i < 5 ; i ++ )
14      printf( " %d" , starCount[ i ] );
15  printf( "\n" );
16
17
18
```

Q: Why is it better to use for loop?

A: E.g., number of viewers can be a variable instead of a fixed value

Outline

- Arrays
  - Basic concepts
  - Two basic operations:
    - How to create an array
    - How to use (access) an array
  - Precaution and Common mistakes
- **MISC topics and multi-dimensional arrays**
- Example Algorithms using Loops and Arrays

An Example

- Any issue in the following program?

```
1
2
3   int main( void )
4   {
5       int array[ 10 ] = { 1 , 2 , 3 , 4 , 5  , \
6                           6 , 7 , 8 , 9 , 10 } ;
7       int i ;
8
9       for( i = 0 ; i < 10 ; i++ )
10          array[ i ] = i ;
11
12      ...
```

An Example

- Define constant "MAX_SIZE" and use it throughout the program; good for *readability* and *program maintenance*, cause you just need to modify 20 in one single line.

```
1   const int MAX_SIZE = 10 ;
2
3   int main( void )
4   {
5       int array[ MAX_SIZE ] = { 1 , 2 , 3 , 4 , 5  , \
6                                 6 , 7 , 8 , 9 , 10 } ;
7       int i ;
8
9       for( i = 0 ; i < MAX_SIZE ; i++ )
10          array[ i ] = i ;
11
12      ...
```

Line continuation (symbol \) allows us to split a long statement into multiple lines for **readability**, so you don't need to scroll left and right to see the whole statement

Note: It is a good practice to define constants in your program to avoid having many "magic numbers in your code. In the future, you will learn "#define"

Misc. Topic #1 – Array Copying

- "**`array1 = array2`**" is **NOT** copying an array!

```
1  int main( void )
2  {
3      int source[ 3 ] = { 1 , 2 , 3 } ;
4      int destination[ 3 ] ;
5      int i ;
6
7      destination = source ;
8
9
10     return 0 ;
11 }
```

Wrong!!!

It is about "pointer" – the starting memory address of the array…
A concept that we will visit in great detail near the end of this course

Misc. Topic #1 – Array Copying

- "**`array1 = array2`**" is **NOT** copying an array!

```
1  int main( void )
2  {
3      int source[ 3 ] = { 1 , 2 , 3 } ;
4      int destination[ 3 ] ;
5      int i ;
6
7      for( i = 0 ; i < 3 ; i++ )
8          destination[ i ] = source[ i ];
9
10     return 0 ;
11 }
```

Correct!!!

Element-by-element copying!

It is about "pointer" – the starting memory address of the array…
A concept that we will visit in great detail near the end of this course

Misc. Topic #2 – Array Comparison

- "**`array1 == array2`**" is **NOT** comparing 2 arrays!

```
 1  int main( void )
 2  {
 3      int array1[ 3 ] = { 1 , 2 , 3 } ;
 4      int array2[ 3 ] = { 3 , 2 , 1 } ;
 5      int same        =   1 ;              // assume "true"
 6      int i ;
 7
 8      if ( array1 == array2 )
 9          same = 1 ;
10      else
11          same = 0 ;
12
13      printf( "Are they the same? %d\n" , same );
14      return 0 ;
15  }
```

Wrong!!!

It is about "pointer" – the starting memory address of the array…
A concept that we will visit in great detail near the end of this course

Misc. Topic #2 – Array Comparison

- "**`array1 == array2`**" is **NOT** comparing 2 arrays!

```
1  int main( void )
2  {
3      int array1[ 3 ] = { 1 , 2 , 3 } ;
4      int array2[ 3 ] = { 3 , 2 , 1 } ;
5      int same        =   1 ;              // assume "true"
6      int i ;
7
8      for( i = 0 ; i < 3 ; i++ )
9      {
10         if ( array1[ i ] != array2[ i ] )
11             same = 0 ;
12     }
13     printf( "Are they the same? %d\n" , same );
14     return 0 ;
15 }
```

Correct!!!

Element-by-element comparison!

Can you speed up?

It is about "pointer" – the starting memory address of the array…
A concept that we will visit in great detail near the end of this course

Misc. Topic #2 – Array Comparison

- "**`array1 == array2`**" is **NOT** comparing 2 arrays!

```
1  int main( void )
2  {
3      int array1[ 3 ] = { 1 , 2 , 3 } ;
4      int array2[ 3 ] = { 3 , 2 , 1 } ;
5      int same        =   1 ;              // assume "true"
6      int i ;
7
8      for( i = 0 ; i < 3 && same != 0 ; i++ )
9      {
10         if ( array1[ i ] != array2[ i ] )
11             same = 0 ;
12     }
13     printf( "Are they the same? %d\n" , same );
14     return 0 ;
15 }
```

Early exit by using test condition

Misc. Topic #2 – Array Comparison

- "**`array1 == array2`**" is **NOT** comparing 2 arrays!

```
1  int main( void )
2  {
3      int array1[ 3 ] = { 1 , 2 , 3 } ;
4      int array2[ 3 ] = { 3 , 2 , 1 } ;
5      int same        =   1 ;              // assume "true"
6      int i ;
7
8      for( i = 0 ; i < 3 ; i++ )
9          if ( array1[ i ] != array2[ i ] )
10         {
11             same = 0 ;
12             break ;
13         }
14     printf( "Are they the same? %d\n" , same );
15     return 0 ;
16 }
```

Or by using break

Misc. Topic #2 – Array Comparison

- "**`array1 == array2`**" is **NOT** comparing 2 arrays!

```
 1  int main( void )
 2  {
 3      int array1[ 3 ] = { 1 , 2 , 3 } ;
 4      int array2[ 3 ] = { 3 , 2 , 1 } ;
 5      //int same        =   1 ;              // avoid "same"
 6      int i ;
 7
 8      for( i = 0 ; i < 3 ; i++ )
 9          if ( array1[ i ] != array2[ i ] )
10              break ;
11
12
13
14      printf( "Are they the same? %d\n" , (i==3) );
15      return 0 ;
16  }
```

Misc. Topic #3 – 2D array

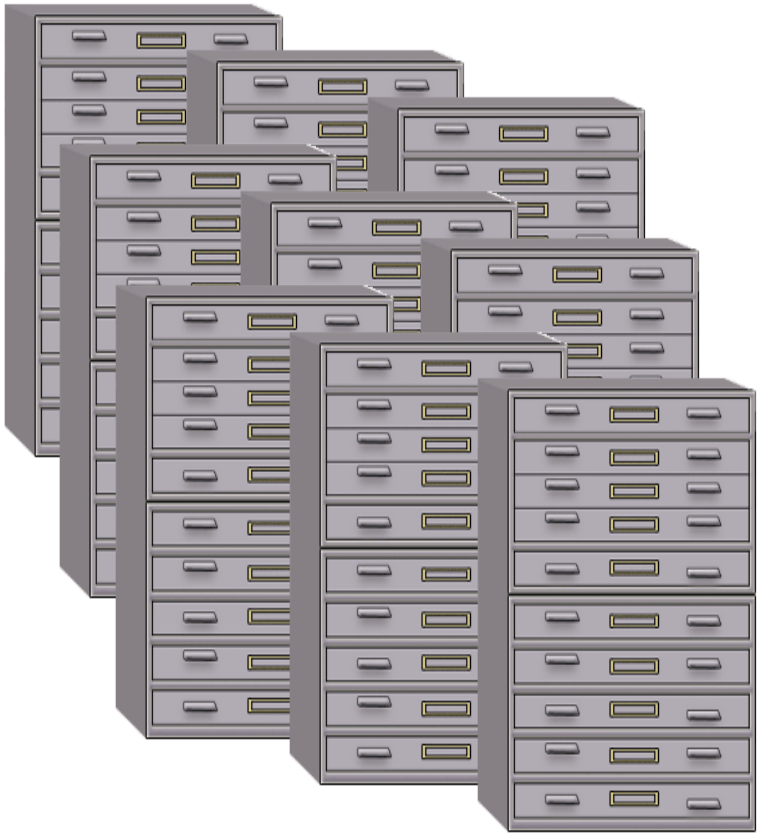
- The array you just learnt is also known as an one-dimensional (1D) array:

```
int data[ 3 ];
```

Index	0	1	2
Value	data[0]	data[1]	data[2]

How we need 2D array?

- 1D array – We can store scores of 50 students
- How to store scores of 50 students for all 9 labs?



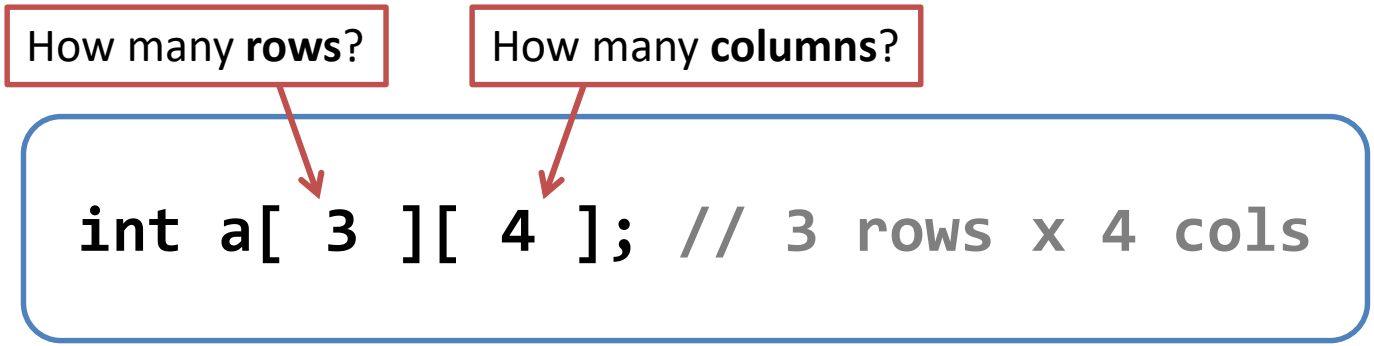
```
// nine arrays!!!
int scores_lab1[50] ;
int scores_lab2[50] ;
…
int scores_lab9[50] ;
```

```
// 2D array -> all scores here!!!
int scores[9][50]   // OR [50][9] ;
```

Better organization of the data!!!

Misc. Topic #3 – 2D array

- There are **multi-dimensional** arrays: 2D, 3D, etc.



Row first
and
then Column

row \ column	0	1	2	3
0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Note: you will use 2D arrays extensively in your project!!!

Misc. Topic #3 – 2D array

- Another Example:

```
int a[ 4 ][ 3 ]; // 4 rows x 3 cols
```

row \ column	0	1	2
0	a[0][0]	a[0][1]	a[0][2]
1	a[1][0]	a[1][1]	a[1][2]
2	a[2][0]	a[2][1]	a[2][2]
3	a[3][0]	a[3][1]	a[3][2]

Row first and then Column

Misc. Topic #3 – 2D array

- Loop through a 2D array:

```
1  int main( void )
2  {
3      int a[ 4 ][ 3 ];
4      int i , j ;
5
6      for( i = 0 ; i < 4 ; i++ ) {
7          for( j = 0 ; j < 3 ; j++ ) {
8              a[ i ][ j ] = i - j ;
9          }
10     }
11     return 0 ;
12 }
```

What are the values stored in "a"?

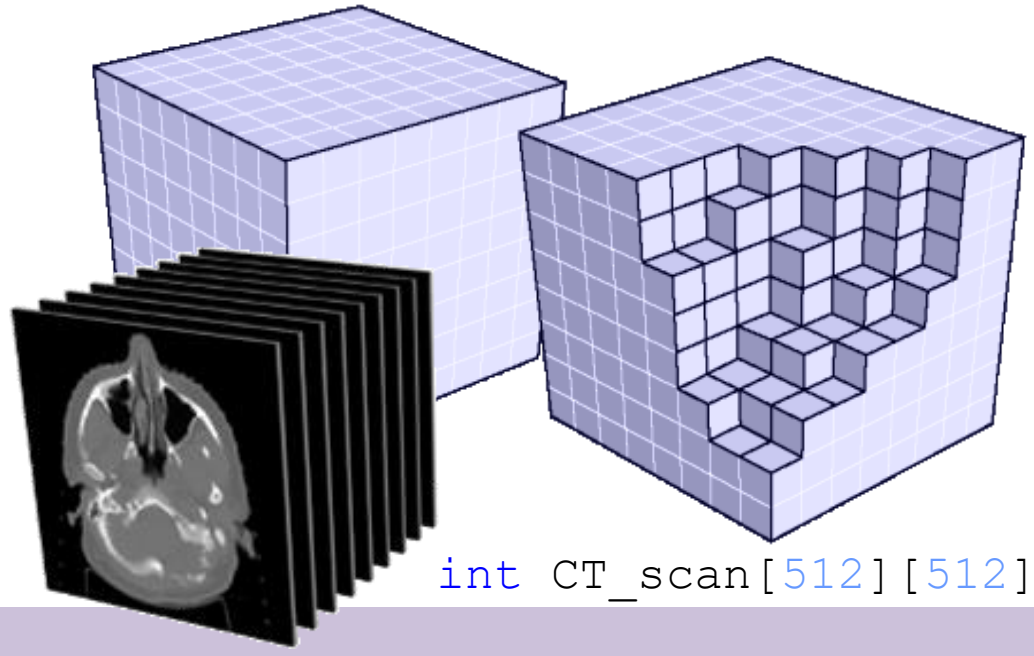
You may write a program and try to print out the values…

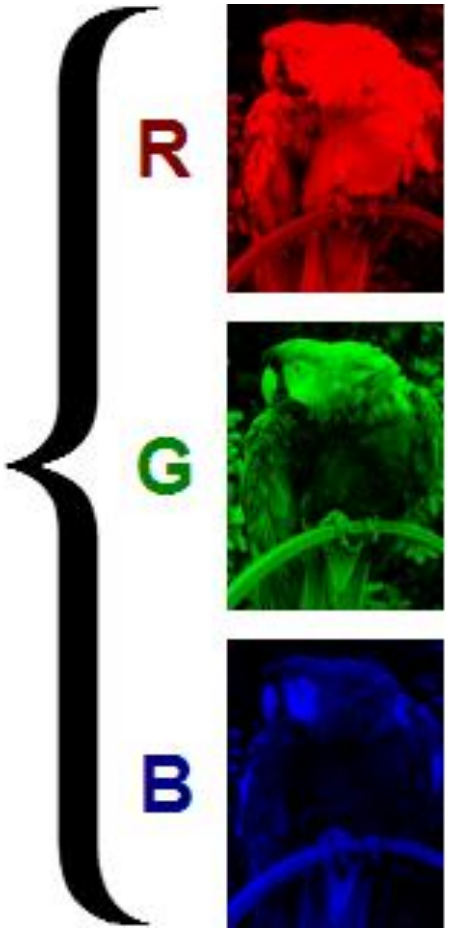
1) Go to the first row (**`i = 0`**)
2) Then, inside the first row, initialize every element, based on the number of columns, i.e., from **`j = 0`** to **`j = "number of columns - 1"`**
3) So and so for

Summary: multi-dimensional array

- We usually use **nested for-loops** to access multi-dimensional arrays.

```
// e.g., an RGB image is a
//       3D array of intensity values
int threeDimArr[1024][2048][3];
```

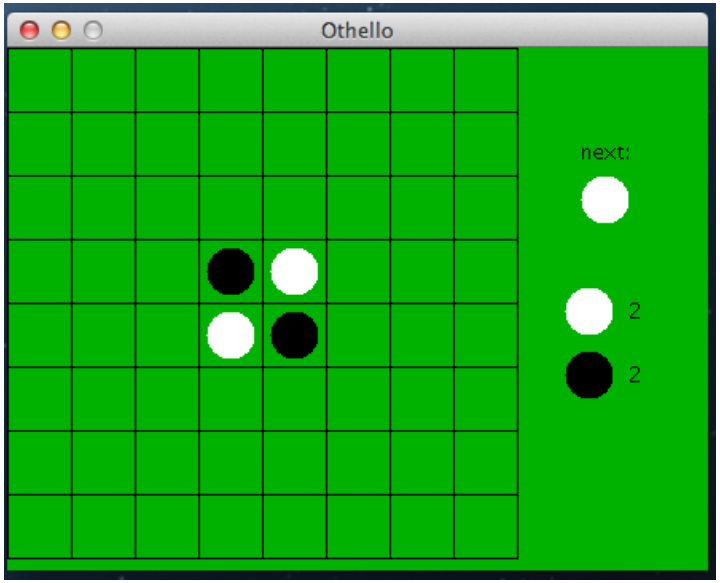


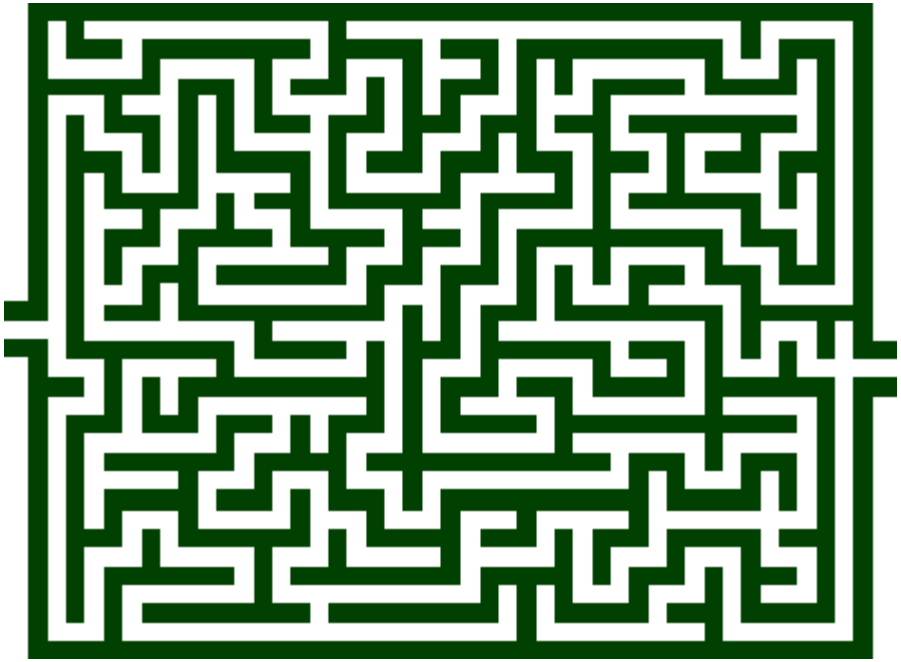



```
int CT_scan[512][512][512];
```

Misc. Topic #3 – 2D array

- Fun with 2D array?
  - Spreadsheet program,
  - Matrix operations,
  - Games, etc.





Outline

- Arrays
  - Basic concepts
  - Two basic operations:
    - How to create an array
    - How to use (access) an array
  - Precaution and Common mistakes
- MISC topics and multi-dimensional arrays
- **Example Algorithms using Loops and Arrays**

Test for Prime Number

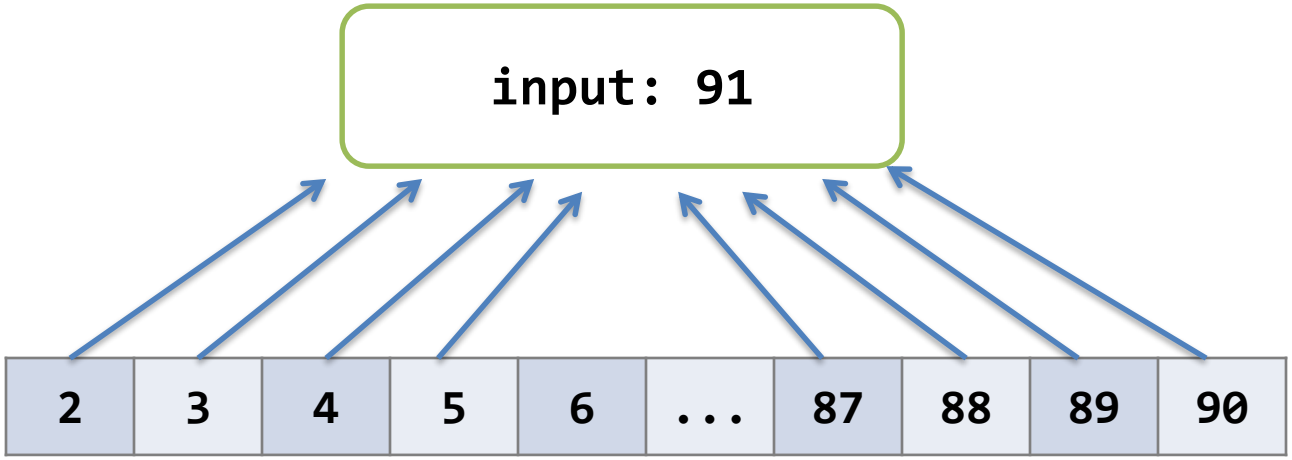
- Prime number? **only divisible by 1 and itself**

  E.g, 2, 3, 5, 7, ……

- **Task**: check if the input is a prime number or not?

Test for Prime Number

- Brute-force approach!



To test if 91 is divisible by any number from 2 to 90:

- If yes, 91 is not a prime number.
- If no, 91 is a prime number.

ENGG2440 will teach you more theoretic way to test for prime.

Test for Prime Solution Code

```
1  int main( void )
2  {
3      unsigned int input , i ;
4      scanf( "%u" , & input );
5
6      for ( i = 2 ; i < input ; i++ ) {
7          if( input % i == 0 )
8              break ;
9      }
10     if ( i == input )  // loop till the end
11         printf( "YES\n" );
12     else
13         printf( "NO\n" );
14     return 0 ;
15 }
```

For the input 4294967291, it takes my new Macbook Pro **10.1 seconds** to say "YES".

Test for Prime Solution Code

- If an integer N is a multiple of A, then there exists integer B such that

```
N = A * B, where B >= 1
```

- Suppose A < B:
  - As A increases, B decreases.
  - Eventually, A >= B!

For "Test for Prime", we don't need to check over those numbers!

Let N = 18

A	B
1	18
2	9
3	6
6	3
9	2
18	1

Test for Prime Solution Code

- To reduce the number of iterations, we can test:

`Given N = A * B, whether A <= N / A?`

- **`A <= N/A?`**
  - Same as asking **`if A <= B`**
  - Ask yourself, **`will A == B?`**

For "Test for Prime", we don't need to check over these numbers!

Let N = 18

A	B	A < N / A?
1	18	YES
2	9	YES
3	6	YES
6	3	NO
9	2	NO
18	1	NO

Test for Prime Solution Code - FASTER

```
1  int main( void )
2  {
3      unsigned int input , i ;
4      scanf( "%u" , & input );
5
6
7      for ( i = 2 ; i <= input / i ; i++ ) {
8          if( input % i == 0 )
9              break ;
10     }
11
12     if ( i > input / i )  // loop till the end
13         printf( "YES\n" );
14     else
15         printf( "NO\n" );
16     return 0 ;
17 }
```

Test up to what value?

THE CHANGE

THE CHANGE

For the input 4294967291, it takes **almost zero seconds** to say "YES"!!!!!!

Test for Prime Solution Code - FASTER

```
1  int main( void )
2  {
3      unsigned int input , i , sqrtN ;
4      scanf( "%u" , & input );
5
6      sqrtN = (unsigned int) ( sqrt( input ) + 0.5 ) ;
7      for ( i = 2 ; i <= sqrtN ; i++ ) {
8          if( input % i == 0 )
9              break ;
10     }
11
12     if ( i > sqrtN )  // loop till the end
13         printf( "YES\n" );
14     else
15         printf( "NO\n" );
16     return 0 ;
17 }
```

Test up to what value?

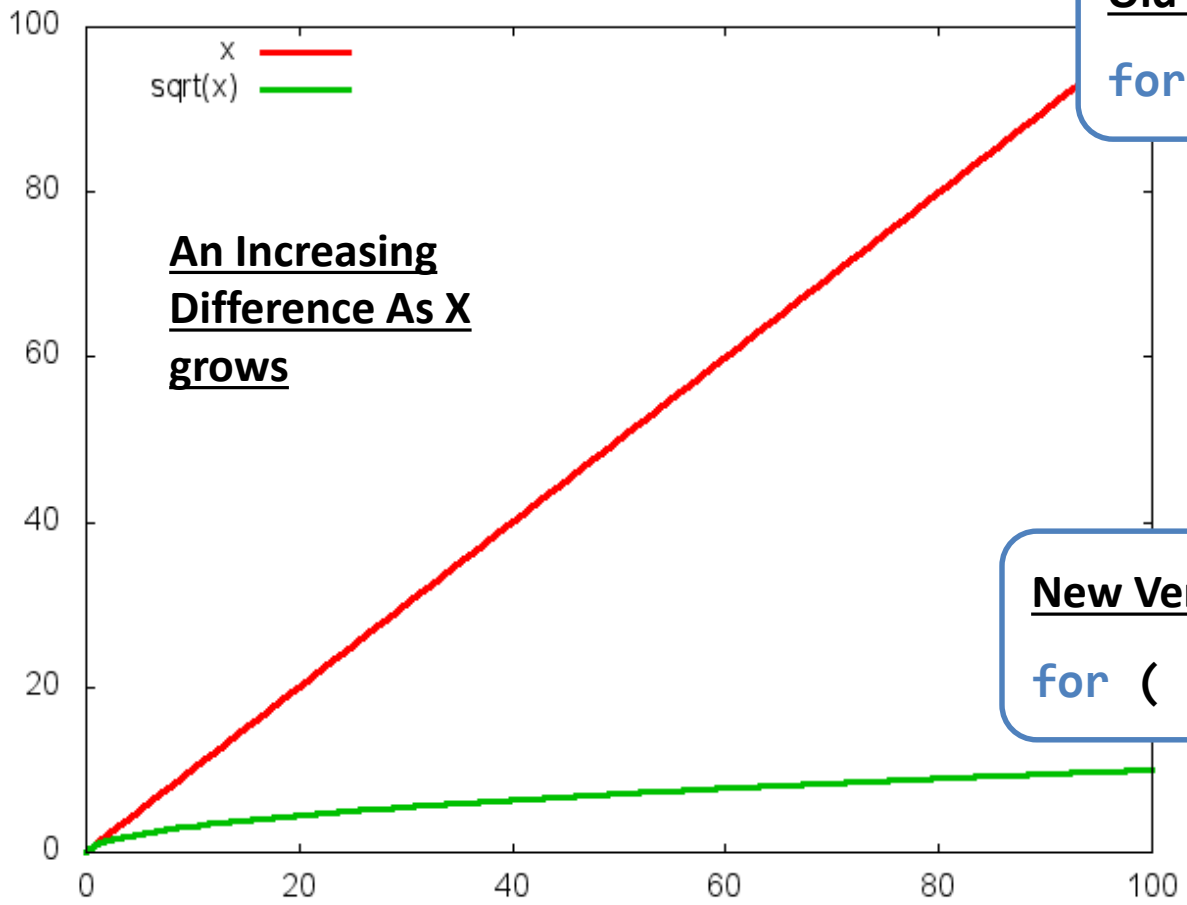
CHANGE

THE CHANGE

Note: we can precompute and store the value in sqrtN

For the input 4294967291, it takes **almost zero seconds** to say "YES"!!!!!!

Note: Efficiency Comparison



Old Version ➔ Loops ≈ N

```
for ( i = 2 ; i < N ; i++ )
```

New Version ➔ Loops ≈ sqrt(N)

```
for ( i = 2 ; i <= N/i ; i++ )
```

Note: Why need + 0.5?

```
6  sqrtN = (unsigned int) ( sqrt( input ) + 0.5 ) ;
7      for ( i = 2 ; i <= sqrtN ; i++ ) ...
```

- Let's try the following code:

```
int N , cubicRootN ;
scanf("%d", &N);
cubicRootN = (int)(pow((double)N, (double)(1.0/3.0)));
printf("cubicRootN = %d\n",cubicRootN);
cubicRootN = (int)(pow((double)N, (double)(1.0/3.0)) + 0.5);
printf("cubicRootN = %d\n",cubicRootN);
```

Input:

```
1000000
```

Your Output:

```
cubicRootN = 99
cubicRootN = 100
```

Next Problem: More on prime numbers

Count how many prime numbers <= N, say 5,000,000?

```
1  int main( void )
2  {
3      int input , i , counter = 0 ;
4
5      for ( input = 2 ; input <= 5000000 ; input++ )
6      {
7          for ( i = 2 ; i <= input / i ; i++ )
8              if( input % i == 0 )
9                  break ;
10         if ( i > input / i )  // loop till the end
11             counter++ ;
12     }
13     printf( "%d\n" , counter );
14
15     return 0 ;
16 }
```

Copied from previous code

Answer: 348513
Time: 2.2905 sec

Next Problem: More on prime numbers

Count how many prime numbers <= N, say 5,000,000?

```
1  int main( void )
2  {
3      int input , i , sqrtN , counter = 0 ;
4
5      for ( input = 2 ; input <= 5000000 ; input++ )
6      {
7          sqrtN = (unsigned int) ( sqrt( input ) + 0.5 ) ;
8          for ( i = 2 ; i <= sqrtN ; i++ )
9              if( input % i == 0 )
10                 break ;
11         if ( i > sqrtN )  // loop till the end
12             counter++ ;
13     }
14     printf( "%d\n" , counter );
15
16     return 0 ;
17 }
```

Copied from previous code

Answer: 348513
Time: 1.1577 sec

Sieve of Eratosthenes

- Watch a nice video first:
  - https://www.youtube.com/watch?v=V08g_lkKj6Q
- This technique (actually an algorithm) was invented by a Greek Mathematician, Eratosthenes, in 240BC!

Sieve of Eratosthenes

- How to implement it?
  - An 1D array, let's call it "**`sieve`**", and
  - Nested loop structure

Initialization Step (1)

`sieve[i] == 1` **means** **`i is a prime number`**, **and vice versa.**
Except 1, for all elements in the array, set it to be 1 at the beginning.

Index	1	2	3	4	5	6	7	8	9	10
Value	0	1	1	1	1	1	1	1	1	1

Sieve of Eratosthenes

- How to implement it?
  - An 1D array, let's call it "**sieve**", and
  - Nested loop structure

Step (2) - Start with 2: `find all i that is a multiple of 2`
➔ `These i must not be a prime number`
➔ `Set sieve[i] = 0`

Index	1	2	3	4	5	6	7	8	9	10
Value	0	1	1	~~1~~ 0	1	~~1~~ 0	1	~~1~~ 0	1	~~1~~ 0

Sieve of Eratosthenes

- How to implement it?
  - An 1D array, let's call it "**`sieve`**", and
  - Nested loop structure

Step (3) - For each k in the array, `where sieve[k] == 1`
➔ `Find all i that is a multiple of k`
➔ `Set sieve[i] = 0`

Index	1	2	3	4	5	6	7	8	9	10
Value	0	1	1	0	1	0	1	0	~~1~~ 0	0

Sieve of Eratosthenes

- How to implement it?
  - An 1D array, let's call it "**`sieve`**", and
  - Nested loop structure

Step (4)
After we are done, for each k in the array, `where sieve[k] == 1`
➔`k is a prime number!`

Index	1	2	3	4	5	6	7	8	9	10
Value	0	1	1	0	1	0	1	0	0	0

Sieve of Eratosthenes – Coding

```
1  int main( void )
2  {
3      int  i , j , counter = 0 ;
4      char sieve[ 5000000 + 1 ] ;  // char saves memory!
5      for ( i = 2 ; i <= 5000000 ; i++ )
6          sieve[ i ] = 1 ;
7
8      for ( i = 2 ; i <= 5000000 ; i++ )
9          if ( sieve[ i ] == 1 )
10         {
11             for ( j = i+1 ; j <= 5000000 ; j++ )
12                 if ( j % i == 0 )
13                     sieve[ j ] = 0 ;
14             counter++ ;
15         }
16
17     printf( "%d\n" , counter );
18     return 0 ;
19 }
```

Initialization

For each prime number, cross out her multiples!

Any more redundant computation?

Answer: ??????
Time: > 30 min.

Sieve of Eratosthenes – Coding

```
1  int main( void )
2  {
3      int  i , j , counter = 0 ;
4      char sieve[ 5000000 + 1 ] ;  // char saves memory!
5      for ( i = 2 ; i <= 5000000 ; i++ )
6          sieve[ i ] = 1 ;
7
8      for ( i = 2 ; i <= 5000000 ; i++ )
9          if ( sieve[ i ] == 1 )
10         {
11             for ( j = i+i ; j <= 5000000 ; j += i )
12                 if ( j % i == 0 )
13                     sieve[ j ] = 0 ;
14             counter++ ;
15         }
16
17     printf( "%d\n" , counter );
18     return 0 ;
19 }
```

Possible to make it even faster?
Note the loop range?

No need to do the testing!!!

CHANGE

Answer: 348513
Time: 0.063 sec

Sieve of Eratosthenes – Coding

```
1  int main( void )
2  {
3      int  i , j , sqrtN , counter = 0 ;
4      char sieve[ 5000000 + 1 ] ;
5      memset( sieve , 1 , 5000001 );
6
7      sqrtN = (unsigned int) ( sqrt(5000000) + 0.5 );
8      for ( i = 2 ; i <= sqrtN ; i++ )
9          if ( sieve[ i ] == 1 )
10         {
11             for ( j = i+i ; j <= 5000000 ; j += i )
12                 sieve[ j ] = 0 ;
13             counter++ ;
14         }
15     for ( ; i <= 5000000 ; i++ )
16         if ( sieve[i] == 1 ) counter++ ;
17
18     printf( "%d\n" , counter );
19     return 0 ;
20 }
```

Using **sqrt** to stop the loop earlier…

Learn yourself:
memset() in C
- For “char” only

CHANGE

CHANGE

Answer: 348513
Time: 0.047 sec

Sieve of Eratosthenes – Exercise

- Let's try: Facebook Hacker Cup 2015, Round 1

  Name: Homework

  https://www.facebook.com/notes/facebook-hacker-cup/hacker-cup-2015-round-1-solutions/1047761065239794/

  – **Goal**: count how many numbers in the range of [A, B] have **C distinct prime factors**, e.g., C=2 for 6, 10, 12, 14, 15, etc.
    - where A, B, and C are inputs to the question

  – Indirect use of the Sieve!
    - **Class Discussion**:
    - How to use the Sieve to count the number of distinct prime factors?