ESC101N
Fundamentals of Computing

Arnab Bhattacharya
arnabb@iitk.ac.in

Indian Institute of Technology, Kanpur
http://www.iitk.ac.in/esc101/

1st semester, 2010-11

Tue, Wed, Fri 0800-0900 at L7

Arrays

- Arrays are lists of uniform variables
- An array is declared using `[]`
- Example

```
int a[5];
```

  represents an array of 5 integers
- Name of the array is `a`
- Individual elements of the array are `a[0]`, `a[1]`, `a[2]`, `a[3]` and `a[4]`
  - Important: Array indexing starts with 0
  - If array `a` has n elements, there is no `a[n]`
  - Error with any index < 0 or $\geq n$
- Why are arrays required?

Arrays

- Arrays are lists of uniform variables
- An array is declared using `[]`
- Example

```
int a[5];
```

  represents an array of 5 integers
- Name of the array is `a`
- Individual elements of the array are `a[0]`, `a[1]`, `a[2]`, `a[3]` and `a[4]`
  - Important: Array indexing starts with 0
  - If array `a` has n elements, there is no `a[n]`
  - Error with any index < 0 or $\geq n$
- Why are arrays required?
- Convenient way to uniformly process variables
- Short name for a collection of variables
- Imagine declaring 100 integers instead of `int a[100];`

Fibonacci series: version 1

- List the first n Fibonacci numbers
- Successively compute the ratio of two adjacent numbers

Fibonacci series: version 1

- List the first n Fibonacci numbers
- Successively compute the ratio of two adjacent numbers

```
#include <stdio.h>
#include <math.h>
int main()
{
    int a[40];
    int i, n;
    double ratio[40];

    printf("Enter number of terms (<= 40): ");
    scanf("%d", &n);

    a[0] = 1; a[1] = 1;
    ratio[0] = 1; ratio[1] = 1;
    for (i = 2; i < n; i++)
    {
        a[i] = a[i - 1] + a[i - 2];
        ratio[i] = a[i] / a[i - 1];
    }

    for (i = 0; i < n; i++)
        printf("%d ", a[i]);
    printf("\n");
    for (i = 0; i < n; i++)
        printf("%lf ", ratio[i]);
    printf("\n");
}
```

- What happened to the ratio?

Fibonacci series: version 2

- / was interpreted as integer division
- Must use *explicit* type casting to `double`

Fibonacci series: version 2

- / was interpreted as integer division
- Must use *explicit* type casting to double

```
#include <stdio.h>
#include <math.h>
int main()
{
    int a[40];
    int i, n;
    double ratio[40];

    printf("Enter number of terms (<= 40): ");
    scanf("%d", &n);

    a[0] = 1; a[1] = 1;
    ratio[0] = 1; ratio[1] = 1;
    for (i = 2; i < n; i++)
    {
        a[i] = a[i - 1] + a[i - 2];
        ratio[i] = (double)a[i] / a[i - 1];
    }

    for (i = 0; i < n; i++)
        printf("%d ", a[i]);
    printf("\n");
    for (i = 0; i < n; i++)
        printf("%lf ", ratio[i]);
    printf("\n");
    printf("Golden ratio = %lf\n", ((1 + sqrt(5)) / 2));
}
```

Nested loop: Seive of Eratosthenes: version 1

- Print all primes less than a specified number

Nested loop: Seive of Eratosthenes: version 1

- Print all primes less than a specified number

```
#include <stdio.h>
#include <math.h>
int main()
{
    int i, j, n;
    int primes[1000];    // to indicate booleans

    printf("Enter number (< 1000): ");
    scanf("%d", &n);

    for (i = 0; i < n; i++)
        primes[i] = 1;  // assume all are primes to start with
    primes[0] = 0; primes[1] = 0;    // 0 and 1 are special cases

    for (i = 2; i < (int)(sqrt(n) + 1); i++)
        for (j = 2; j < n; j++)
            if ((j != i) && ((j % i) == 0)) // check for factors
                primes[j] = 0;  // j has a factor and is not a prime

    for (i = 0; i < n; i++)
        if (primes[i] == 1) // print only primes
            printf("%d ", i);
    printf("\n");
}
```

Nested loop: Seive of Eratosthenes: version 1

- Print all primes less than a specified number

```
#include <stdio.h>
#include <math.h>
int main()
{
    int i, j, n;
    int primes[1000];    // to indicate booleans

    printf("Enter number (< 1000): ");
    scanf("%d", &n);

    for (i = 0; i < n; i++)
        primes[i] = 1;  // assume all are primes to start with
    primes[0] = 0; primes[1] = 0;    // 0 and 1 are special cases

    for (i = 2; i < (int)(sqrt(n) + 1); i++)
        for (j = 2; j < n; j++)
            if ((j != i) && ((j % i) == 0)) // check for factors
                primes[j] = 0;  // j has a factor and is not a prime

    for (i = 0; i < n; i++)
        if (primes[i] == 1) // print only primes
            printf("%d ", i);
    printf("\n");
}
```

- Outer loop uses `i` ($\sqrt{n}$ times)
- In each iteration of outer loop, `i` is fixed for inner loop (n times)
- Inner loop uses `j` along with this fixed `i` (total of $n\sqrt{n}$ times)

Seive of Eratosthenes: version 2

- Checking `j` up to `i` is unnecessary as there cannot be any factor of `i` before it

Seive of Eratosthenes: version 2

- Checking `j` up to `i` is unnecessary as there cannot be any factor of `i` before it

```
#include <stdio.h>
#include <math.h>
int main()
{
    int i, j, n;
    int primes[1000];    // to indicate booleans

    printf("Enter number (< 1000): ");
    scanf("%d", &n);

    for (i = 0; i < n; i++)
        primes[i] = 1;  // assume all are primes to start with
    primes[0] = 0; primes[1] = 0;   // 0 and 1 are special cases

    for (i = 2; i < (int)(sqrt(n) + 1); i++)
        for (j = i + 1; j < n; j++)
            if ((j % i) == 0) // check for factors
                primes[j] = 0;  // j has a factor and is not a prime

    for (i = 0; i < n; i++)
        if (primes[i] == 1) // print only primes
            printf("%d ", i);
    printf("\n");
}
```

- Checking whether `j == i` is now meaningless

Seive of Eratosthenes: version 3

- If a number is already composite (i.e., non-prime), there is no need to re-check its multiples

Seive of Eratosthenes: version 3

- If a number is already composite (i.e., non-prime), there is no need to re-check its multiples

```
#include <stdio.h>
#include <math.h>
int main()
{
    int i, j, n;
    int primes[1000];    // to indicate booleans

    printf("Enter number (< 1000): ");
    scanf("%d", &n);

    for (i = 0; i < n; i++)
        primes[i] = 1;  // assume all are primes to start with
    primes[0] = 0; primes[1] = 0;   // 0 and 1 are special cases

    for (i = 2; i < (int)(sqrt(n) + 1); i++)
    {
        if (primes[i] == 0) // no need to re-check a composite number
            continue;
        for (j = i + 1; j < n; j++)
            if ((j % i) == 0) // check for factors
                primes[j] = 0;  // j has a factor and is not a prime
    }

    for (i = 0; i < n; i++)
        if (primes[i] == 1) // print only primes
            printf("%d ", i);
    printf("\n");
}
```

Seive of Eratosthenes: version 4

- Starting from `i + 1` is unnecessary as the next factor is `2 * i`

Seive of Eratosthenes: version 4

- Starting from `i + 1` is unnecessary as the next factor is `2 * i`

```c
#include <stdio.h>
#include <math.h>
int main()
{
    int i, j, n;
    int primes[1000];    // to indicate booleans

    printf("Enter number (< 1000): ");
    scanf("%d", &n);

    for (i = 0; i < n; i++)
        primes[i] = 1;  // assume all are primes to start with
    primes[0] = 0; primes[1] = 0;    // 0 and 1 are special cases

    for (i = 2; i < (int)(sqrt(n) + 1); i++)
    {
        if (primes[i] == 0) // no need to re-check a composite number
            continue;
        for (j = 2 * i; j < n; j++)
            if ((j % i) == 0) // check for factors
                primes[j] = 0;  // j has a factor and is not a prime
    }

    for (i = 0; i < n; i++)
        if (primes[i] == 1) // print only primes
            printf("%d ", i);
    printf("\n");
}
```

Seive of Eratosthenes: version 5

- Since only multiples of `i` are needed, `j` can be increased in steps of `i`

Seive of Eratosthenes: version 5

- Since only multiples of `i` are needed, `j` can be increased in steps of `i`

```
#include <stdio.h>
#include <math.h>
int main()
{
    int i, j, n;
    int primes[1000];    // to indicate booleans

    printf("Enter number (< 1000): ");
    scanf("%d", &n);

    for (i = 0; i < n; i++)
        primes[i] = 1;  // assume all are primes to start with
    primes[0] = 0; primes[1] = 0;    // 0 and 1 are special cases

    for (i = 2; i < (int)(sqrt(n) + 1); i++)
    {
        if (primes[i] == 0) // no need to re-check a composite number
            continue;
        for (j = 2 * i; j < n; j = j + i)
            primes[j] = 0;  // j has a factor and is not a prime
    }

    for (i = 0; i < n; i++)
        if (primes[i] == 1) // print only primes
            printf("%d ", i);
    printf("\n");
}
```

- `j` is now guaranteed to be a multiple of `i` and checking whether `(j % i) == 0` is unnecessary

Seive of Eratosthenes: version 6

- `j` can start from `i * i` as every other multiple has already been checked by an earlier `j`

Seive of Eratosthenes: version 6

- `j` can start from `i * i` as every other multiple has already been checked by an earlier `j`

```c
#include <stdio.h>
#include <math.h>
int main()
{
    int i, j, n;
    int primes[1000];    // to indicate booleans

    printf("Enter number (< 1000): ");
    scanf("%d", &n);

    for (i = 0; i < n; i++)
        primes[i] = 1;  // assume all are primes to start with
    primes[0] = 0; primes[1] = 0;    // 0 and 1 are special cases

    for (i = 2; i < (int)(sqrt(n) + 1); i++)
    {
        if (primes[i] == 0) // no need to re-check a composite number
            continue;
        for (j = i * i; j < n; j = j + i)
            primes[j] = 0;  // j has a factor and is not a prime
    }

    for (i = 0; i < n; i++)
        if (primes[i] == 1) // print only primes
            printf("%d ", i);
    printf("\n");
}
```

Two-dimensional arrays

- Arrays can be of arrays
- They form two-dimensional arrays
- A two-dimensional array is declared using `[] []`
- Example

```
double a[2][3];
```

  represents a (2-element) array of (3-element) array of doubles
- Name of the array is `a`
- Individual elements of `a` are `a[0]` to `a[2]` which are in turn arrays themselves
- Individual `double` values are `a[0][0]`, `a[0][1]`, etc. up to `a[1][2]`
- Total number of elements is $2 \times 3 = 6$

Two-dimensional arrays

- Arrays can be of arrays
- They form two-dimensional arrays
- A two-dimensional array is declared using `[] []`
- Example

```
double a[2][3];
```

  represents a (2-element) array of (3-element) array of doubles
- Name of the array is `a`
- Individual elements of `a` are `a[0]` to `a[2]` which are in turn arrays themselves
- Individual `double` values are `a[0][0]`, `a[0][1]`, etc. up to `a[1][2]`
- Total number of elements is $2 \times 3 = 6$
- Convenient way to describe matrices

Two-dimensional arrays

- Arrays can be of arrays
- They form two-dimensional arrays
- A two-dimensional array is declared using `[] []`
- Example

```
double a[2][3];
```

  represents a (2-element) array of (3-element) array of doubles
- Name of the array is `a`
- Individual elements of `a` are `a[0]` to `a[2]` which are in turn arrays themselves
- Individual `double` values are `a[0][0]`, `a[0][1]`, etc. up to `a[1][2]`
- Total number of elements is $2 \times 3 = 6$
- Convenient way to describe matrices
- Multi-dimensional arrays can be declared in the same way

```
double a[3][2][4];
```

Matrix reading and writing

- To print a matrix in the correct format, newlines (`\n`) are required

Matrix reading and writing

- To print a matrix in the correct format, newlines (\n) are required

```
#include <stdio.h>
int main()
{
    double m[3][3];
    int i, j;

    for (i = 0; i < 3; i++)
        for (j = 0; j < 3; j++)
            scanf("%lf", &m[i][j]);

    printf("The input matrix is:\n");
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 3; j++)
            printf("%lf\t", m[i][j]);
        printf("\n");
    }
}
```

- Tab characters (\t) are used to align the columns

Transpose of a matrix

- To print the transpose of a matrix, interchange the indices

Transpose of a matrix

- To print the transpose of a matrix, interchange the indices

```c
#include <stdio.h>
int main()
{
    double m[3][3];
    int i, j;

    for (i = 0; i < 3; i++)
        for (j = 0; j < 3; j++)
            scanf("%lf", &m[i][j]);

    printf("The transposed matrix is:\n");
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 3; j++)
            printf("%lf\t", m[j][i]);
        printf("\n");
    }
}
```

Transpose of a matrix

- To print the transpose of a matrix, interchange the indices

```c
#include <stdio.h>
int main()
{
    double m[3][3];
    int i, j;

    for (i = 0; i < 3; i++)
        for (j = 0; j < 3; j++)
            scanf("%lf", &m[i][j]);

    printf("The transposed matrix is:\n");
    for (i = 0; i < 3; i++)
    {
        for (j = 0; j < 3; j++)
            printf("%lf\t", m[j][i]);
        printf("\n");
    }
}
```

- Works only for square matrices

Transpose of a matrix: wrong version

- For a general matrix, interchanging the indices will not work

Transpose of a matrix: wrong version

- For a general matrix, interchanging the indices will not work

```
#include <stdio.h>
int main()
{
    double m[2][3];
    int i, j;

    for (i = 0; i < 2; i++)
        for (j = 0; j < 3; j++)
            scanf("%lf", &m[i][j]);

    printf("The input matrix is:\n");
    for (i = 0; i < 2; i++)
    {
        for (j = 0; j < 3; j++)
            printf("%lf\t", m[i][j]);
        printf("\n");
    }

    printf("The transposed matrix is:\n");
    for (i = 0; i < 2; i++)
    {
        for (j = 0; j < 3; j++)
            printf("%lf\t", m[j][i]);
        printf("\n");
    }
}
```

Transpose of a matrix: wrong version

- For a general matrix, interchanging the indices will not work

```
#include <stdio.h>
int main()
{
    double m[2][3];
    int i, j;

    for (i = 0; i < 2; i++)
        for (j = 0; j < 3; j++)
            scanf("%lf", &m[i][j]);

    printf("The input matrix is:\n");
    for (i = 0; i < 2; i++)
    {
        for (j = 0; j < 3; j++)
            printf("%lf\t", m[i][j]);
        printf("\n");
    }

    printf("The transposed matrix is:\n");
    for (i = 0; i < 2; i++)
    {
        for (j = 0; j < 3; j++)
            printf("%lf\t", m[j][i]);
        printf("\n");
    }
}
```

- There is no element corresponding to `m[2][0]`
- The element `m[0][2]` is never printed

Transpose of a matrix: correct version

- Interchanging the loops works
  - Indices should *not* be interchanged

Transpose of a matrix: correct version

- Interchanging the loops works
  - Indices should *not* be interchanged

```
#include <stdio.h>
int main()
{
    double m[2][3];
    int i, j;

    for (i = 0; i < 2; i++)
        for (j = 0; j < 3; j++)
            scanf("%lf", &m[i][j]);

    printf("The input matrix is:\n");
    for (i = 0; i < 2; i++)
    {
        for (j = 0; j < 3; j++)
            printf("%lf\t", m[i][j]);
        printf("\n");
    }

    printf("The transposed matrix is:\n");
    for (j = 0; j < 3; j++)
    {
        for (i = 0; i < 2; i++)
            printf("%lf\t", m[i][j]);
        printf("\n");
    }
}
```