CS1100 – Introduction to Programming

Instructor:

Shweta Agrawal (shweta.a@cse.iitm.ac.in)

Lecture 17

Character arrays

char name[20];

Different ways of initialization

- char name[20] = “Avani”;
- char name[20] = {‘A’, ‘V’, ‘A’, ‘N’, ‘I’, ’null char’};
- char name[20];
  scanf(“%s”, name);

Character arrays

char name[20];

Different ways of initialization

- char name[20] = “Avani”;
- char name[20] = {‘A’, ‘V’, ‘A’, ‘N’, ‘I’, ’null char’};
- char name[20];
  scanf(“%s”, name);
- char name[20];
  name = ”AVANI”; Incorrect!!

What is the output of this program?

```
#include<stdio.h>
main() {
    char name[20] = "AVANI";
    int i;

    for (i=10; i<20; i++) {
        name[i] = 'X';
    }

    printf("name = %s\n", name);

    for (i=0; i<20; i++) {
        printf("%c %d\n", name[i], name[i]);
    }
}
```

Character arrays and standard library support

- Character arrays or strings occur very often.
- C provides a standard library string.h
- exposes several useful functions:
  - strlen
  - strcmp
  - strcpy
  - strstr
- But we can create libraries.

Example 1 : Finding the length of a given string

Task : Given a string at the input, find the length.

Example 1 : Finding the length of a given string

Task : Given a string at the input, find the length.

Pseudo-code :

for i ranging from 1 to n

- if you find that i^{th} character is null character output i and break.

Example 1 : Finding the length of a given string

Task : Given a string at the input, find the length.

Pseudo-code :

for i ranging from 1 to n

- if you find that i^{th} character is null character output i and break.

Program Segment:

```
char s[1000], i;
scanf("%s", s);

for(i=0 ; s[i] != '\0'; ++i);

printf("Length : %d", i);
```

Example 2 : Compare two strings

Task : Given two strings $s1, s2$, check if $s1$ and $s2$ are the same.

Example 2 : Compare two strings

Task : Given two strings $s1, s2$, check if $s1$ and $s2$ are the same.

if (s1 == s2)

This does not work

Example 2 : Compare two strings

Task : Given two strings $s1, s2$, check if $s1$ and $s2$ are the same.

if (s1 == s2)

This does not work

Pseudo-code :

Find the length ℓ of the two strings first. If they are different, declare that the strings are different.

For i ranging from 1 to ℓ

- check if i^{th} characters are the same. If not, declare NOT same.

Can we combine the two steps above?

Example 2 : Compare two strings

Task : Given two strings $s1, s2$, check if $s1$ and $s2$ are the same.
if (s1 == s2)

This does not work

Pseudo-code :

Find the length ℓ of the two strings first. If they are different, declare that the strings are different.

For i ranging from 1 to ℓ

- check if i^{th} characters are the same. If not, declare NOT same.

Can we combine the two steps above?

Program Segment:

```
// strings are in arrays a and b

int i = 0;

 while (a[i] == b[i]){
    if ((a[i] == '\0')||(b[i] == '\0')
       break;
    i++;
 }

 if (a[i] == '\0' && b[i] == '\0')
    printf("SAME");
 else
    printf("NOT SAME");
```

Example 3 : Palindromes

A string is a palindrome iff string == reverse(string)

Example 3 : Palindromes

A string is a palindrome iff string == reverse(string)

- malayalam
- neveroddoreven
- dontnod

Example 3 : Palindromes

A string is a palindrome iff string == reverse(string)

- malayalam
- neveroddoreven
- dontnod

Write a program to determine if the given string is a palindrome.

Example 3 : Palindromes

Task : Given a string check if it is a palindrome.

Pseudo-code :

- Run and index i from 1 to n and another j from n to 1.
- check if i^{th} character is equal to j^{th} character. If not, declare NOT PALINDROME.
- If all checks pass - then declare PALINDROME.
- You can do better

Example 3 : Palindromes

Task : Given a string check if it is a palindrome.

Pseudo-code :

- Run and index i from 1 to n and another j from n to 1.
- check if i^{th} character is equal to j^{th} character. If not, declare NOT PALINDROME.
- If all checks pass - then declare PALINDROME.
- You can do better

Program Segment:

```
    // string is in the array named str
    int l = 0;
    int h = strlen(str) - 1;

    while (h > l)
    {
        if (str[l++] != str[h--])
        {
            printf("NOT PALINDROME");
            break;
        }
    }
if (h == l)
  printf("PALINDROME");
    // spot the error
```

Multi-dimensional arrays in C

- Declaring a multi-dimensional array
  int myArray[size1][size2]...[sizeN];
  int matrix [10][10];

Multi-dimensional arrays in C

- Declaring a multi-dimensional array
  int myArray[size1][size2]...[sizeN];
  int matrix [10][10];
- How is a two-dimensional array stored in memory?

Multi-dimensional arrays in C

- Declaring a multi-dimensional array
  int myArray[size1][size2]...[sizeN];
  int matrix [10][10];
- How is a two-dimensional array stored in memory?
- Initializing a two-dimensional array.

Multi-dimensional arrays in C

- Declaring a multi-dimensional array
  int myArray[size1][size2]...[sizeN];
  int matrix [10][10];
- How is a two-dimensional array stored in memory?
- Initializing a two-dimensional array.

```
#include<stdio.h>

main() {
    int matrix[3][4] = {
        {1, 2, 3, 4},
        {5, 6, 7, 8},
        {9, 10, 11, 12}
    }
}
```

Multi-dimensional arrays in C

- Accessing elements of the array : `A[i][j]` - element in row i and column j of array A.

Multi-dimensional arrays in C

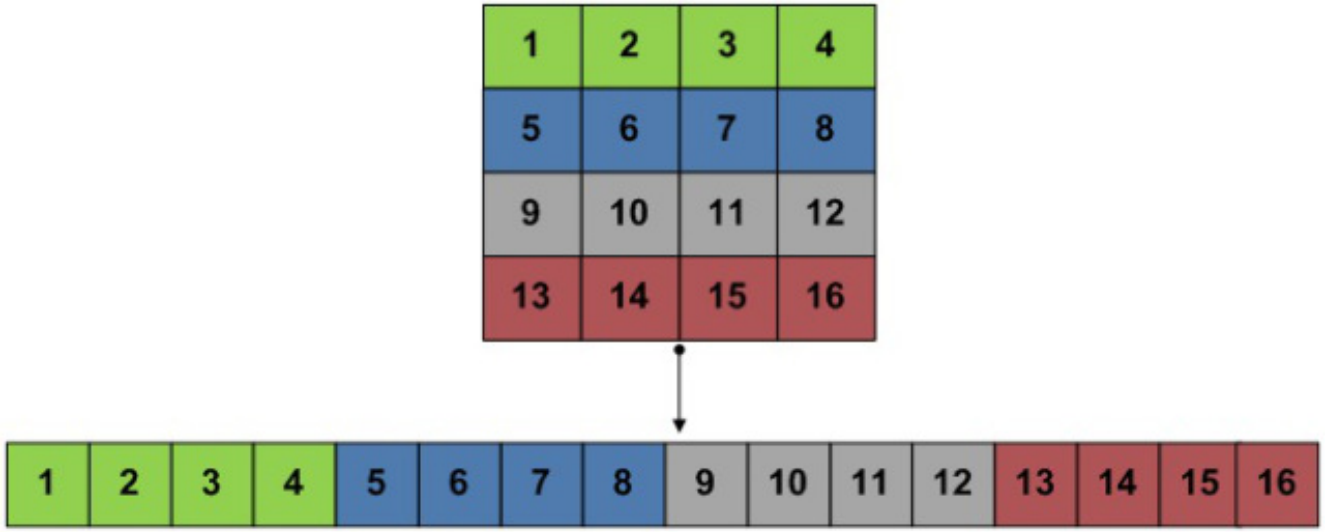
- Accessing elements of the array : `A[i][j]` - element in row i and column j of array A.
- Rows/columns numbered from 0.

Multi-dimensional arrays in C

- Accessing elements of the array : `A[i][j]` - element in row i and column j of array A.
- Rows/columns numbered from 0.
- Storage: row-major ordering elements of row 0, elements of row 1, etc.

Multi-dimensional arrays in C

- Accessing elements of the array : `A[i][j]` - element in row i and column j of array A.
- Rows/columns numbered from 0.
- Storage: row-major ordering elements of row 0, elements of row 1, etc.



Initializing Multi-dimensional arrays

```
#include<stdio.h>

main() {
    int matrix1[3][4] = {
        1, 2, 3, 4,
        5, 6, 7, 8,
        9, 10, 11, 12};


    int matrix2[][4] = {
        1, 2, 3, 4,
        5, 6, 7, 8,
        9, 10, 11, 12};
}
```

Initializing Multi-dimensional arrays

```
#include<stdio.h>

main() {
    int matrix1[3][4] = {
        1, 2, 3, 4,
        5, 6, 7, 8,
        9, 10, 11, 12};


    int matrix2[][4] = {
        1, 2, 3, 4,
        5, 6, 7, 8,
        9, 10, 11, 12};
}
```

- Cannot omit the column size.

Initializing Multi-dimensional arrays

What does the program print?

```
/* Assume N1=3, N2=4 */
main() {
    int matrix[N1][N2] = {
        1, 2, 3, 4,
        5, 6, 7, 8,
        9, 10, 11, 12};

    int i;
    for (i = 0; i<N1; i++) {
        printf("%d\n", matrix[i][2]++);
    }

    for (i = 0; i<N2; i++) {
        printf("%d\t", matrix[2][i]);
    }
    printf("\n");
}
```

Reading/Writing matrices at the input

`mat` : name of the matrix

`rows`, `cols` : number of rows and columns.

Reading/Writing matrices at the input

`mat` : name of the matrix

`rows`, `cols` : number of rows and columns.

Reading Matrices from the input:

```
for (int i = 0; i < rows; i++)
  for (int j = 0; j < cols; j++)
      scanf("%d", &mat[i][j]);
```

Reading/Writing matrices at the input

`mat` : name of the matrix

`rows`, `cols` : number of rows and columns.

Reading Matrices from the input:

```
for (int i = 0; i < rows; i++)
  for (int j = 0; j < cols; j++)
      scanf("%d", &mat[i][j]);
```

Writing matrices to the output:

```
for (int i = 0; i < rows; i++)
{
  for (int j = 0; j < cols;  j++)    /* print a row */
     {  printf ("%d ", mat[i][j]); }    /* notice missing \n */
  printf ("\n");            /* print a newline at the end a row */
}
```

Matrix Operations : Addition

- Write a program to add two matrices A and B

Matrix Operations : Addition

- Write a program to add two matrices A and B

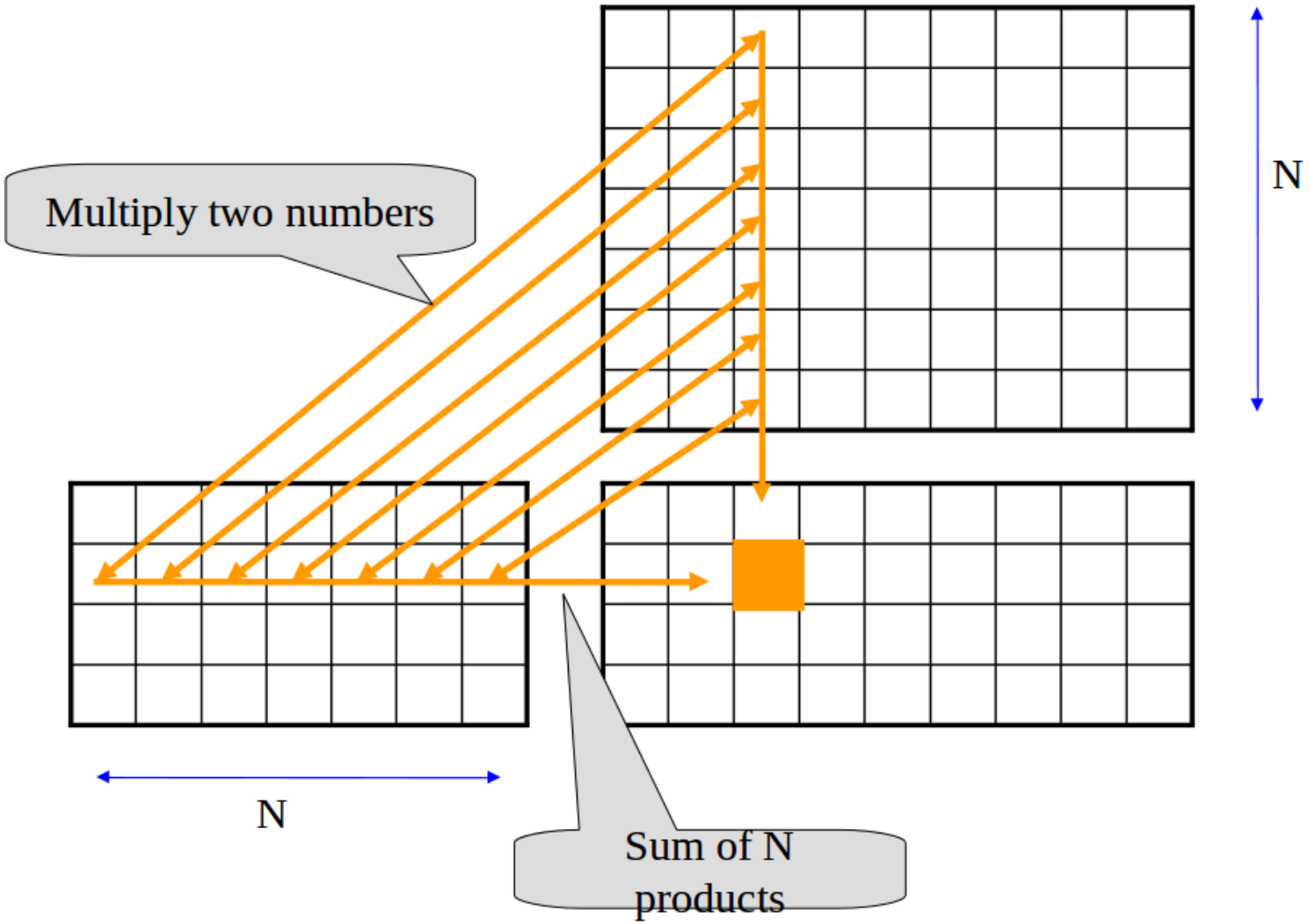
```
#include<stdio.h>
main() {

    /* Assume N1 and N2 are defined as const int */

    int A[N1][N2];
    int B[N1][N2];
    /*initialize M1, M2 suitably */
    int C[N1][N2];
    int i, j;

    for (i = 0; i<N1; i++) {
        for (j = 0; j<N2; j++) {
            A[i][j] = B[i][j] + C[i][j];
        }
    }
}
```

Matrix Operations : Multiplication

- Write a program to multiply matrices A and B

Matrix Operations : Multiplication

- Write a program to multiply matrices A and B



Matrix Operations : Multiplication

- Write a program to multiply matrices A and B

Matrix Operations : Multiplication

- Write a program to multiply matrices A and B

```
int main() {

    const int N1;
    int A[N1][N1], B[N1][N1], C[N1][N1];
    int i, j, k, sum;
    /* Assume A, B are initialized suitably */

    for (i = 0; i<N1; i++) {
        for (j = 0; j<N1; j++) {
            sum = 0;
            for (k=0; k<N1; k++) {
                /* fill in your code here */
            }
            C[i][j] = sum;
        }
    }
}
```

Summary

- **Technical:** Learned about single, multidimensional arrays, character arrays, strings. Declaration, Initialization, reading, writing.
- **Problem Solving :** Writing programs to solve various tasks associated where use of arrays, matrices, strings are natural.

Summary

- **Technical:** Learned about single, multidimensional arrays, character arrays, strings. Declaration, Initialization, reading, writing.
- **Problem Solving :** Writing programs to solve various tasks associated where use of arrays, matrices, strings are natural.
- **Meta-level message about the approach :** Writing algorithms/pseudo-code/programs - identify simpler tasks within the given task, solve them and and then try to combine them to get the bigger solution.

Summary

- **Technical:** Learned about single, multidimensional arrays, character arrays, strings. Declaration, Initialization, reading, writing.
- **Problem Solving :** Writing programs to solve various tasks associated where use of arrays, matrices, strings are natural.
- **Meta-level message about the approach :** Writing algorithms/pseudo-code/programs - identify simpler tasks within the given task, solve them and and then try to combine them to get the bigger solution.
- **Observation:** Subtasks that appear once solved, can be used in several parts of the program.

Summary

- **Technical:** Learned about single, multidimensional arrays, character arrays, strings. Declaration, Initialization, reading, writing.
- **Problem Solving :** Writing programs to solve various tasks associated where use of arrays, matrices, strings are natural.
- **Meta-level message about the approach :** Writing algorithms/pseudo-code/programs - identify simpler tasks within the given task, solve them and and then try to combine them to get the bigger solution.
- **Observation:** Subtasks that appear once solved, can be used in several parts of the program. Functions !!