
Week 4

XMUT-NWEN 241 - 2026 T2

Systems Programming

Agatha Rachmat

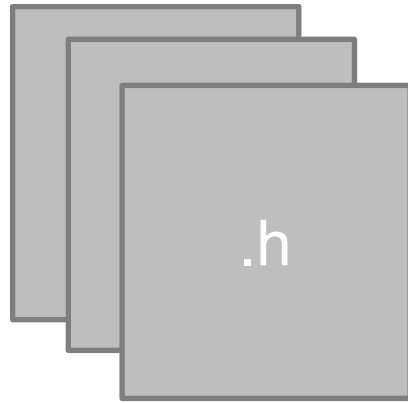
School of Engineering and Computer Science

Victoria University of Wellington

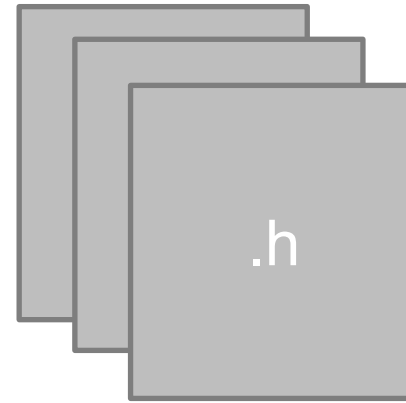
Content

- More on Arrays

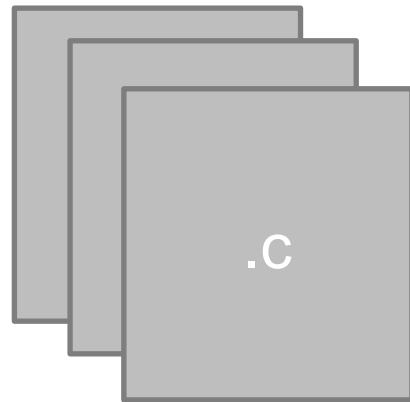
Recall: Large C Program



Header files
from standard
C library



Own header
files



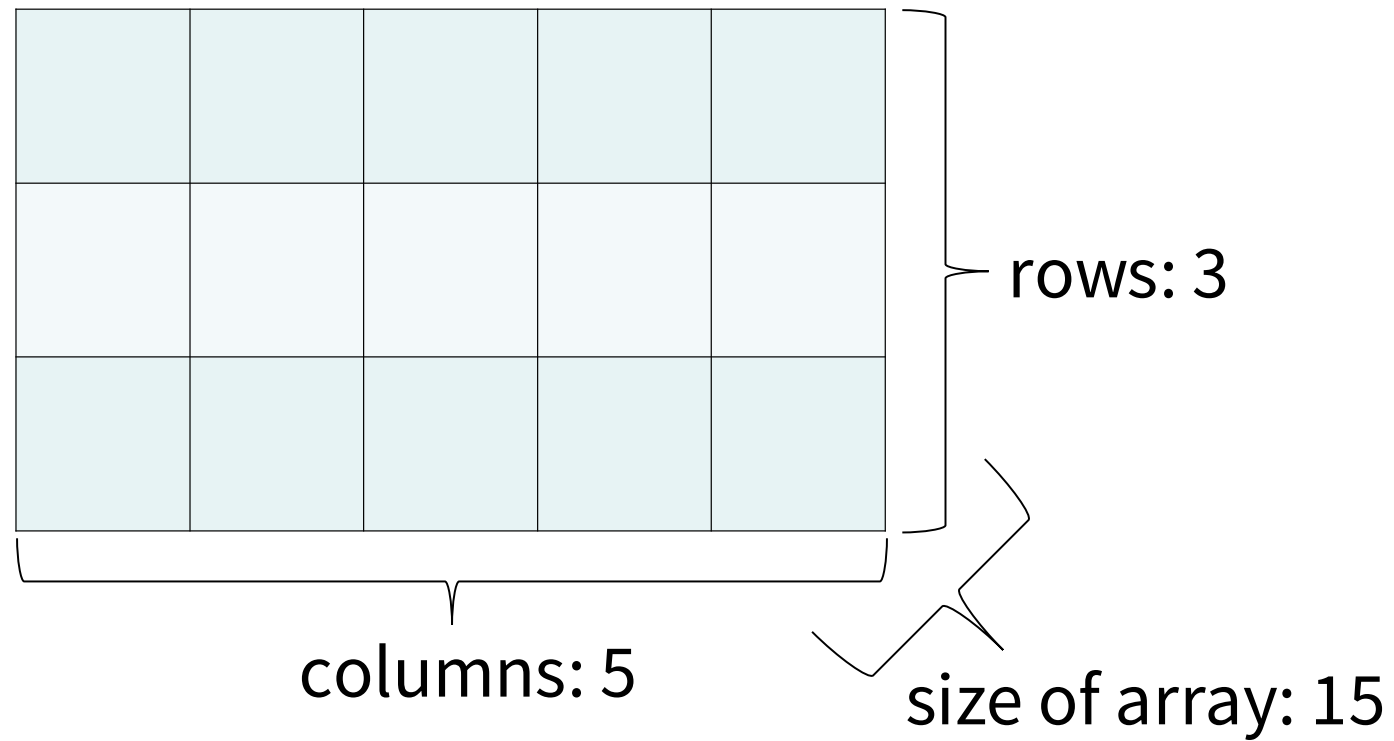
Source files

Multi-dimensional Arrays

Multi-dimensional Arrays

- In C, you can create array of an array known as multidimensional array
- The simplest interpretation of a multi-dimensional array is a table, i.e. a **two-dimensional array**
 - each **row** has the same number of **columns**

Two-Dimensional Arrays Overview (1)



Two-Dimensional Arrays Overview (2)

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15



array of ints

Two-Dimensional Arrays Overview (3)

1.0	2.0	3.0	4.0	5.0
6.0	7.0	8.0	9.0	10.0
11.0	12.0	13.0	14.0	15.0



array of floats

Two-Dimensional Arrays Overview (4)

	0	1	2	3	4
0	H	e	l	l	o
1		W	o	r	l
2	d		?	!	

element at row 3 column 4
`array[2][3]`



array elements

size of array: 15

Two-Dimensional Arrays

- Declaring a char array with 3 rows and 5 columns

```
char two_d[3][5];
```

- The array can hold 15 char elements

- Accessing a value

```
char ch;  
ch = two_d[2][4];
```

- Modifying a value

```
two_d[0][0] = 'x';
```

- The array can be initialized in one of the following ways

```
int two_d[2][3] = {{5, 2, 1}, {6, 7, 8}};  
int two_d[2][3] = {5, 2, 1, 6, 7, 8};  
int two_d[][3] = {{5, 2, 1}, {6, 7, 8}};
```

- The number of columns must be explicitly stated. The compiler will find the appropriate amount of rows based on the initializer list

Passing 2D Arrays to Functions (1)

- Passing a single array element to a function
 - can be passed in a similar manner as passing a variable to a function

```
void display(int a) {  
    printf("%d", a);  
}  
  
int main(void) {  
    int age[2][3] = { {18, 19, 20}, {21, 22, 23} };  
  
    display(age[1][2]); /* Passing element age[1][2] only */  
  
    return 0;  
}
```

Passing 2D Arrays to Functions (2)

- Passing an entire array to a function
 - When passing an array as an argument to a function, it is passed by its memory address (starting address of the memory area) and not its value(**call-by-address**)!
 - Because a function accesses the original array values, we must be very careful that we do not inadvertently (accidentally) change values in an array within a function.

```
void enterData(int d[][10]) {  
    /* Code for reading and saving data into 2D array */  
}  
  
int main(void)  
{  
    int data[10][10];  
  
    enterData(data);  
}
```

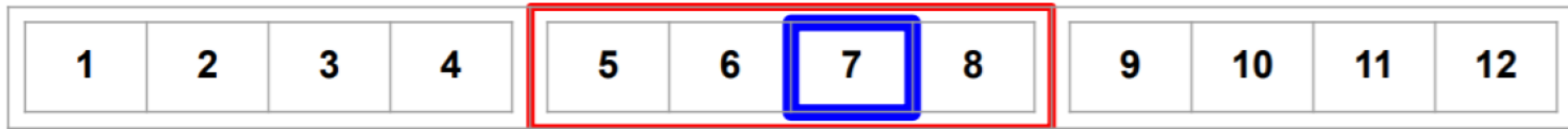
Arrays

- Arrays are second class citizens
- With an array, you can NOT:
 - Change the size after initialization
 - Assign a new array using '='
- In addition, arrays automatically 'decay' into pointers, losing information about their size (with few exceptions).
 - More on array decay (after you learn pointers)!

2D Arrays

- Multi-dimensional arrays are typically contiguous.

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



- They also need additional information to index into the correct position. When passed to a function for example, it needs to know how many values to 'skip' to get to an inner array.

Matrix Multiplication Basics

- Given two matrices **A** (of size $m \times n$) and **B** (of size $n \times p$), the product **C** (of size $m \times p$) is calculated as:

$$C_{ij} = \sum_{k=1}^n A_{ik} \times B_{kj}$$

- Where:
- C_{ij} is the element in the i th row and j th column of matrix **C**.
- A_{ik} is the element in the i th row and k th column of matrix **A**.
- B_{kj} is the element in the k th row and j th column of matrix **B**.
- Important: The number of columns in **A** must equal the number of rows in **B**.

Matrix Multiplication Basics

- Given two matrices **A** (of size $m \times n$) and **B** (of size $n \times p$), the product **C** (of size $m \times p$) is calculated as:

$$C_{ij} = \sum_{k=1}^n A_{ik} \times B_{kj}$$

- Where:
- C_{ij} is the element in the i^{th} row and j^{th} column of matrix C.
- A_{ik} is the element in the i^{th} row and k^{th} column of matrix A.
- B_{kj} is the element in the k^{th} row and j^{th} column of matrix B.
- Important: The number of columns in A must equal the number of rows in B.

Example: Multi dimension Array

Type	Example	Real-world use
1D	int marks[5]	Marks of one student
2D	int marks[10][5]	Students × subjects
2D	int seats[20][10]	Cinema rows × seats
2D	int sales[5][12]	Products × months
3D	int data[2][3][4]	Warehouses × shelves × products
3D	int attendance[2][30][7]	Classes × students × days

Example: Multi dimension Array

1. Student Marks: Store marks of 3 students in 4 subjects.

marks[student][subject]

```
#include <stdio.h>

int main() {
    int marks[3][4] = {
        {85, 90, 78, 88},
        {72, 80, 91, 75},
        {95, 89, 92, 96}
    };
    for (int i = 0; i < 3; i++) {
        printf("Student %d: ", i + 1);
        for (int j = 0; j < 4; j++) {
            printf("%d ", marks[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

Example: Multi dimension Array

2. Cinema Seating

- First dimension = **row**
- Second dimension = **seat number**

Screen

Example: Multi dimension Array

```
#include <stdio.h>

int main() {
    int seats[4][5] = {
        {1, 0, 1, 0, 0},
        {0, 0, 1, 1, 0},
        {1, 1, 0, 0, 0},
        {0, 1, 0, 1, 1}
    };
    printf("Cinema Seating:\n");
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 5; j++) {
            if (seats[i][j] == 1)
                printf("[X] ");
            else
                printf("[ ] ");
        }
        printf("\n");
    }
    return 0;
}
```

0 = available
1 = occupied.

- First dimension = **row**
- Second dimension = **seat number**

Output:

```
[X] [ ] [X] [ ] [ ]
[ ] [ ] [X] [X] [ ]
[X] [X] [ ] [ ] [ ]
[ ] [X] [ ] [X] [X]
```

Example: Multi dimension Array

3D Array — Student Attendance

Suppose a school has:

- 2 classes
- 3 students in each class
- 5 days of attendance

attendance[class][student][day]

1 = **present**, while 0 = **absent**.

```
#include <stdio.h>

int main() { // [class][student][day]
    int attendance[2][3][5] = {
        {
            {1, 1, 1, 0, 1},
            {1, 0, 1, 1, 1},
            {1, 1, 1, 1, 1}
        },
        {
            {1, 1, 0, 1, 1},
            {0, 1, 1, 1, 0},
            {1, 1, 1, 0, 1}
        }
    };
    printf("Class 1, Student 2, Day 4: %d\n",
        attendance[0][1][3]);
    return 0;
}
```

Review

Write a simple program to print temperature records

Record temperatures for 7 days across 3 different cities.

	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
City 1	20	21	19	22	23	21	20
City 2	25	26	24	27	28	26	25
City 3	15	16	14	17	18	16	15

```
#include <stdio.h>

int main() {
    int temperature[_][_ ] = {_____};

    for (int city = 0; city < 3; city++) {
        _____
    }
    return 0;
}
```

Next Lecture

- Strings