Introduction to Programming

Week – 5, Lecture – 2

Arrays in C – Part 2

SAURABH SRIVASTAVA

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

IIT KANPUR

The arrays that we know till now…

They are collections of variables of the same type

The arrays that we know till now…

They are collections of variables of the same type

To access a particular variable in the array, we use its position, i.e. index within the array

The arrays that we know till now…

They are collections of variables of the same type

To access a particular variable in the array, we use its position, i.e. index within the array

What we have used till now, are actually *one-dimensional* arrays

The arrays that we know till now…

They are collections of variables of the same type

To access a particular variable in the array, we use its position, i.e. index within the array

What we have used till now, are actually *one-dimensional* arrays

It means, if we want to visualise them, we can see them like “variables in a line”

- Remember that we used an array to score marks for a student
- An array of size 5, essentially stored marks for 5 courses

The arrays that we know till now…

They are collections of variables of the same type

To access a particular variable in the array, we use its position, i.e. index within the array

What we have used till now, are actually *one-dimensional* arrays

It means, if we want to visualise them, we can see them like “variables in a line”

- Remember that we used an array to score marks for a student
- An array of size 5, essentially stored marks for 5 courses

What if, we have to store marks for more students than one?

- One way to do so would be to increase the size of the array… in multiples of 5

The arrays that we know till now…

They are collections of variables of the same type

To access a particular variable in the array, we use its position, i.e. index within the array

What we have used till now, are actually *one-dimensional* arrays

It means, if we want to visualise them, we can see them like "variables in a line"

- Remember that we used an array to score marks for a student
- An array of size 5, essentially stored marks for 5 courses

What if, we have to store marks for more students than one?

- One way to do so would be to increase the size of the array… in multiples of 5
- The first 5 indices could be used to store marks for the first student
- The next 5 indices could be used to store marks for the second student… and so on

Arrays and memory

There is something special about the variables that are part of an array

Arrays and memory

There is something special about the variables that are part of an array

They are all allotted memory, right next to each other

Arrays and memory

There is something special about the variables that are part of an array

They are all allotted memory, right next to each other

- The term we use is “contiguous memory allocation”, meaning one unbroken block of memory
- This is true, irrespective of the type of the array

Arrays and memory

There is something special about the variables that are part of an array

They are all allotted memory, right next to each other

- The term we use is “contiguous memory allocation”, meaning one unbroken block of memory
- This is true, irrespective of the type of the array
- Thus, an `int` array of size 5 is allocated a block of 20 bytes (4 bytes X 5) …
- … and a `double` array of the same size is allocated a block of 40 bytes (8 bytes X 5)

Arrays and memory

There is something special about the variables that are part of an array

They are all allotted memory, right next to each other

- The term we use is “contiguous memory allocation”, meaning one unbroken block of memory
- This is true, irrespective of the type of the array
- Thus, an `int` array of size 5 is allocated a block of 20 bytes (4 bytes X 5) …
- … and a `double` array of the same size is allocated a block of 40 bytes (8 bytes X 5)

If we know the starting address and the type of an array…

- …we can find addresses of variables at different indices by simple arithmetic

Arrays and memory

There is something special about the variables that are part of an array

They are all allotted memory, right next to each other

- The term we use is “contiguous memory allocation”, meaning one unbroken block of memory
- This is true, irrespective of the type of the array
- Thus, an `int` array of size 5 is allocated a block of 20 bytes (4 bytes X 5) in…
- … and a `double` array of the same size is allocated a block of 40 bytes (8 bytes X 5)

If we know the starting address and the type of an array…

- …we can find addresses of variables at different indices by simple arithmetic

For example, if a C array is declared at starting address `1000` like this: `int arr[5];` then

Arrays and memory

There is something special about the variables that are part of an array

They are all allotted memory, right next to each other

- The term we use is “contiguous memory allocation”, meaning one unbroken block of memory
- This is true, irrespective of the type of the array
- Thus, an `int` array of size 5 is allocated a block of 20 bytes (4 bytes X 5) in…
- … and a `double` array of the same size is allocated a block of 40 bytes (8 bytes X 5)

If we know the starting address and the type of an array…

- …we can find addresses of variables at different indices by simple arithmetic

For example, if a C array is declared at starting address `1000` like this: `int arr[5];` then

- … address of `arr[0]` = `1000`, address of `arr[1]` = `1000` + `4` = `1004`…
- … address of `arr[2]` = `1000` + `8` = `1008`, and so on…

Arrays and memory

There is something special about the variables that are part of an array

They are all allotted memory, right next to each other

- The term we use is “contiguous memory allocation”, meaning one unbroken block of memory
- This is true, irrespective of the type of the array
- Thus, an `int` array of size 5 is allocated a block of 20 bytes (4 bytes X 5) in…
- … and a `double` array of the same size is allocated a block of 40 bytes (8 bytes X 5)

If we know the starting address and the type of an array…

- …we can find addresses of variables at different indices by simple arithmetic

For example, if a C array is declared at starting address `1000` like this: `int arr[5];` then

- … address of `arr[0]` = `1000`, address of `arr[1]` = `1000` + `4` = `1004`…
- … address of `arr[2]` = `1000` + `8` = `1008`, and so on…
- **Basically, address of `arr[i]` = starting address of `arr` + (`i` * size of single element of `arr`)**

Adding more “dimensions” to an array

Let us go back to the issue of storing students’ marks

Adding more “dimensions” to an array

Let us go back to the issue of storing students’ marks

- We can store marks for *N* students by creating an array with *N * 5* elements…

Adding more “dimensions” to an array

Let us go back to the issue of storing students’ marks

- We can store marks for *N* students by creating an array with *N * 5* elements…
- … the marks of the first student is stored in indices `0` to `4`…
- … the marks of the second student is stored in indices `5` to `9`… and so on…

Adding more “dimensions” to an array

Let us go back to the issue of storing students’ marks

- We can store marks for *N* students by creating an array with *N * 5* elements…
- … the marks of the first student is stored in indices `0` to `4`…
- … the marks of the second student is stored in indices `5` to `9`… and so on…
- Basically, for the `i`th student, the marks are stored in indices (`i * 5`) to (`i * 5 + 4`)

Adding more “dimensions” to an array

Let us go back to the issue of storing students’ marks

- We can store marks for *N* students by creating an array with *N * 5* elements…
- … the marks of the first student is stored in indices `0` to `4`…
- … the marks of the second student is stored in indices `5` to `9`… and so on…
- Basically, for the `i`th student, the marks are stored in indices (`i * 5`) to (`i * 5 + 4`)

Arrays provide another way to use the allocated memory block…

- You can “add a dimension” to the array, and use another index, to point to the students…

Adding more “dimensions” to an array

Let us go back to the issue of storing students’ marks

- We can store marks for *N* students by creating an array with *N * 5* elements…
- … the marks of the first student is stored in indices `0` to `4`…
- … the marks of the second student is stored in indices `5` to `9`… and so on…
- Basically, for the `i`th student, the marks are stored in indices (`i * 5`) to (`i * 5 + 4`)

Arrays provide another way to use the allocated memory block…

- You can “add a dimension” to the array, and use another index, to point to the students…
- … so instead of creating `marks[`*`N`*`* 5]` we can also create something like `marks[`*`N`*`][5]`

Adding more “dimensions” to an array

Let us go back to the issue of storing students’ marks

- We can store marks for *N* students by creating an array with *N * 5* elements…
- … the marks of the first student is stored in indices `0` to `4`…
- … the marks of the second student is stored in indices `5` to `9`… and so on…
- Basically, for the `i`th student, the marks are stored in indices (`i * 5`) to (`i * 5 + 4`)

Arrays provide another way to use the allocated memory block…

- You can “add a dimension” to the array, and use another index, to point to the students…
- … so instead of creating `marks[`*N*`* 5]` we can also create something like `marks[`*N*`][5]`
- The amount of memory allocated in both cases is the same, i.e. for (`5 *` *N*) elements

Adding more “dimensions” to an array

Let us go back to the issue of storing students’ marks

- We can store marks for *N* students by creating an array with *N * 5* elements…
- … the marks of the first student is stored in indices `0` to `4`…
- … the marks of the second student is stored in indices `5` to `9`… and so on…
- Basically, for the `i`th student, the marks are stored in indices (`i * 5`) to (`i * 5 + 4`)

Arrays provide another way to use the allocated memory block…

- You can “add a dimension” to the array, and use another index, to point to the students…
- … so instead of creating `marks[`*`N`*`* 5]` we can also create something like `marks[`*`N`*`][5]`
- The amount of memory allocated in both cases is the same, i.e. for (*`N`*`* 5`) elements

One pair of brackets, essentially represent one *dimension* of the array

Adding more “dimensions” to an array

Let us go back to the issue of storing students’ marks

- We can store marks for *N* students by creating an array with *N * 5* elements…
- … the marks of the first student is stored in indices `0` to `4`…
- … the marks of the second student is stored in indices `5` to `9`… and so on…
- Basically, for the `i`th student, the marks are stored in indices (`i * 5`) to (`i * 5 + 4`)

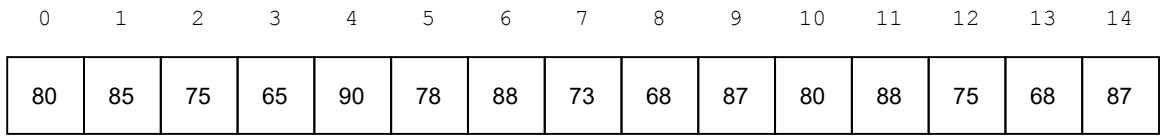
Arrays provide another way to use the allocated memory block…

- You can “add a dimension” to the array, and use another index, to point to the students…
- … so instead of creating `marks[`*`N`*`* 5]` we can also create something like `marks[`*`N`*`][5]`
- The amount of memory allocated in both cases is the same, i.e. for (*`N`*`* 5`) elements

One pair of brackets, essentially represent one *dimension* of the array

- Thus, a 2-dimensional array can be visualised like “variables arranged in rows and columns of a matrix”

Visualising Arrays



A one-dimensional array with the indices,
can be visualized as a line of variables

Visualising Arrays

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
80	85	75	65	90	78	88	73	68	87	80	88	75	68	87

A one-dimensional array with the indices, can be visualized as a line of variables

	0	1	2	3	4
0	80	85	75	65	90
1	78	88	73	68	87
2	80	88	75	68	87

A two-dimensional array with the indices, can be visualized as a rows and columns of variables

Visualising Arrays

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
80	85	75	65	90	78	88	73	68	87	80	88	75	68	87

A one-dimensional array with the indices, can be visualized as a line of variables

This is the first index

	0	1	2	3	4
0	80	85	75	65	90
1	78	88	73	68	87
2	80	88	75	68	87

A two-dimensional array with the indices, can be visualized as a rows and columns of variables

Visualising Arrays

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
80	85	75	65	90	78	88	73	68	87	80	88	75	68	87

A one-dimensional array with the indices, can be visualized as a line of variables

This is the second index

	0	1	2	3	4
0	80	85	75	65	90
1	78	88	73	68	87
2	80	88	75	68	87

A two-dimensional array with the indices, can be visualized as a rows and columns of variables

Visualising Arrays

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
80	85	75	65	90	78	88	73	68	87	80	88	75	68	87

A one-dimensional array with the indices, can be visualized as a line of variables

	0	1	2	3	4
0	80	85	75	65	90
1	78	88	73	68	87
2	80	88	75	68	87

A two-dimensional array with the indices, can be visualized as a rows and columns of variables

In memory, though, they both look alike

1000	80
1004	85
1008	75
1012	65
1016	90
1020	78
1024	88
1028	73
1032	68
1036	87
1040	80
1044	88
1048	75
1052	68
1056	87

Nested loops for multidimensional arrays

Similar to how we usually use a for loop to access all elements of an array…

- … we usually use nested for loops to access all elements of a multidimensional array

Nested loops for multidimensional arrays

Similar to how we usually use a for loop to access all elements of an array…

- … we usually use nested for loops to access all elements of a multidimensional array

For example an array declared like `int arr[M][N];` can be accessed as

```
for(i = 0; i < M; i++)
{
    for(j = 0; j < N; j++)
    {
        printf("%d\t", arr[i][j]);
        if(j == N-1)
            printf("\n");
    }
}
```

Marks summer for multiple students…

```c
int total_marks[N], total_maximum_marks[N];
int marks[N*5];
int max[N*5];
int i;

printf("Marks summer for %d student(s)\n", N);

for(i = 0; i < N*5; i++)
{
        if(i % 5 == 0)
        {
                printf("-------------------------------------------------------\n");
                printf("Enter marks for Student#%d\n", i/5+1); // Integer division ;)
                printf("Please provide marks for five subjects\n");
                printf("Enter the marks in the format obtained/maximum\n");
                printf("Example:\n");
                printf("90/100\n");
                total_marks[i/5] = total_maximum_marks[i/5] = 0;
        }
        scanf("%d/%d", &marks[i], &max[i]);
        total_marks[i/5] += marks[i];
        total_maximum_marks[i/5] += max[i];
}

printf("Here are the sum of marks for all students:\n");

for(i = 0; i < N; i++)
        printf("Student#%d: %d/%d\n", i+1, total_marks[i], total_maximum_marks[i]);
```

```c
int total_marks[N], total_maximum_marks[N];
int marks[N][5];
int max[N][5];
int i, j;

printf("Marks summer for %d student(s)\n", N);

for(i = 0; i < N; i++)
{
        printf("-------------------------------------------------------\n");
        printf("Enter marks for Student#%d\n", i+1);
        printf("Please provide marks for five subjects\n");
        printf("Enter the marks in the format obtained/maximum\n");
        printf("Example:\n");
        printf("90/100\n");
        total_marks[i] = total_maximum_marks[i] = 0;
        for(j = 0; j < 5; j++)
        {
                scanf("%d/%d", &marks[i][j], &max[i][j]);
                total_marks[i] += marks[i][j];
                total_maximum_marks[i] += max[i][j];
        }
}

printf("Here are the sum of marks for all students:\n");

for(i = 0; i < N; i++)
        printf("Student#%d: %d/%d\n", i+1, total_marks[i], total_maximum_marks[i]);
```

Marks summer for multiple students…

```c
int total_marks[N], total_maximum_marks[N];
int marks[N*5];
int max[N*5];
int i;

printf("Marks summer for %d student(s)\n", N);

for(i = 0; i < N*5; i++)
{
        if(i % 5 == 0)
        {
                printf("-------------------------------------------------------\n");
                printf("Enter marks for Student#%d\n", i/5+1); // Integer division ;)
                printf("Please provide marks for five subjects\n");
                printf("Enter the marks in the format obtained/maximum\n");
                printf("Example:\n");
                printf("90/100\n");
                total_marks[i/5] = total_maximum_marks[i/5] = 0;
        }
        scanf("%d/%d", &marks[i], &max[i]);
        total_marks[i/5] += marks[i];
        total_maximum_marks[i/5] += max[i];
}

printf("Here are the sum of marks for all students:\n");

for(i = 0; i < N; i++)
        printf("Student#%d: %d/%d\n", i+1, total_marks[i], total_maximum_marks[i]);
```

```c
int total_marks[N], total_maximum_marks[N];
int marks[N][5];
int max[N][5];
int i, j;

printf("Marks summer for %d student(s)\n", N);

for(i = 0; i < N; i++)
{
        printf("-------------------------------------------------------\n");
        printf("Enter marks for Student#%d\n", i+1);
        printf("Please provide marks for five subjects\n");
        printf("Enter the marks in the format obtained/maximum\n");
        printf("Example:\n");
        printf("90/100\n");
        total_marks[i] = total_maximum_marks[i] = 0;
        for(j = 0; j < 5; j++)
        {
                scanf("%d/%d", &marks[i][j], &max[i][j]);
                total_marks[i] += marks[i][j];
                total_maximum_marks[i] += max[i][j];
        }
}

printf("Here are the sum of marks for all students:\n");

for(i = 0; i < N; i++)
        printf("Student#%d: %d/%d\n", i+1, total_marks[i], total_maximum_marks[i]);
```

Here, we have a 1-dimensional array of size $N * 5$

Marks summer for multiple students…

```
int total_marks[N], total_maximum_marks[N];
int marks[N*5];
int max[N*5];
int i;

printf("Marks summer for %d student(s)\n", N);

for(i = 0; i < N*5; i++)
{
        if(i % 5 == 0)
        {
                printf("-------------------------------------------------------\n");
                printf("Enter marks for Student#%d\n", i/5+1); // Integer division ;)
                printf("Please provide marks for five subjects\n");
                printf("Enter the marks in the format obtained/maximum\n");
                printf("Example:\n");
                printf("90/100\n");
                total_marks[i/5] = total_maximum_marks[i/5] = 0;
        }
        scanf("%d/%d", &marks[i], &max[i]);
        total_marks[i/5] += marks[i];
        total_maximum_marks[i/5] += max[i];
}

printf("Here are the sum of marks for all students:\n");

for(i = 0; i < N; i++)
        printf("Student#%d: %d/%d\n", i+1, total_marks[i], total_maximum_marks[i]);
```

```
int total_marks[N], total_maximum_marks[N];
int marks[N][5];
int max[N][5];
int i, j;

printf("Marks summer for %d student(s)\n", N);

for(i = 0; i < N; i++)
{
        printf("-------------------------------------------------------\n");
        printf("Enter marks for Student#%d\n", i+1);
        printf("Please provide marks for five subjects\n");
        printf("Enter the marks in the format obtained/maximum\n");
        printf("Example:\n");
        printf("90/100\n");
        total_marks[i] = total_maximum_marks[i] = 0;
        for(j = 0; j < 5; j++)
        {
                scanf("%d/%d", &marks[i][j], &max[i][j]);
                total_marks[i] += marks[i][j];
                total_maximum_marks[i] += max[i][j];
        }
}

printf("Here are the sum of marks for all students:\n");

for(i = 0; i < N; i++)
        printf("Student#%d: %d/%d\n", i+1, total_marks[i], total_maximum_marks[i]);
```

and here, we use a 2-dimensional array of size $N \ X \ 5$

Marks summer for multiple students…

```
int total_marks[N], total_maximum_marks[N];
int marks[N*5];
int max[N*5];
int i;

printf("Marks summer for %d student(s)\n", N);

for(i = 0; i < N*5; i++)
{
        if(i % 5 == 0)
        {
                printf("-------------------------------------------------------\n");
                printf("Enter marks for Student#%d\n", i/5+1); // Integer division ;)
                printf("Please provide marks for five subjects\n");
                printf("Enter the marks in the format obtained/maximum\n");
                printf("Example:\n");
                printf("90/100\n");
                total_marks[i/5] = total_maximum_marks[i/5] = 0;
        }
        scanf("%d/%d", &marks[i], &max[i]);
        total_marks[i/5] += marks[i];
        total_maximum_marks[i/5] += max[i];
}

printf("Here are the sum of marks for all students:\n");

for(i = 0; i < N; i++)
        printf("Student#%d: %d/%d\n", i+1, total_marks[i], total_maximum_marks[i]);
```

```
int total_marks[N], total_maximum_marks[N];
int marks[N][5];
int max[N][5];
int i, j;

printf("Marks summer for %d student(s)\n", N);

for(i = 0; i < N; i++)
{
        printf("-------------------------------------------------------\n");
        printf("Enter marks for Student#%d\n", i+1);
        printf("Please provide marks for five subjects\n");
        printf("Enter the marks in the format obtained/maximum\n");
        printf("Example:\n");
        printf("90/100\n");
        total_marks[i] = total_maximum_marks[i] = 0;
        for(j = 0; j < 5; j++)
        {
                scanf("%d/%d", &marks[i][j], &max[i][j]);
                total_marks[i] += marks[i][j];
                total_maximum_marks[i] += max[i][j];
        }
}

printf("Here are the sum of marks for all students:\n");

for(i = 0; i < N; i++)
        printf("Student#%d: %d/%d\n", i+1, total_marks[i], total_maximum_marks[i]);
```

In particular, see the use of a single loop vs two loops with nesting

Marks summer for multiple students…

```
int total_marks[N], total_maximum_marks[N];
int marks[N*5];
int max[N*5];
int i;

printf("Marks summer for %d student(s)\n", N);

for(i = 0; i < N*5; i++)
{
        if(i % 5 == 0)
        {
                printf("-------------------------------------------------------\n");
                printf("Enter marks for Student#%d\n", i/5+1); // Integer division ;)
                printf("Please provide marks for five subjects\n");
                printf("Enter the marks in the format obtained/maximum\n");
                printf("Example:\n");
                printf("90/100\n");
                total_marks[i/5] = total_maximum_marks[i/5] = 0;
        }
        scanf("%d/%d", &marks[i], &max[i]);
        total_marks[i/5] += marks[i];
        total_maximum_marks[i/5] += max[i];
}

printf("Here are the sum of marks for all students:\n");

for(i = 0; i < N; i++)
        printf("Student#%d: %d/%d\n", i+1, total_marks[i], total_maximum_marks[i]);
```

```
int total_marks[N], total_maximum_marks[N];
int marks[N][5];
int max[N][5];
int i, j;

printf("Marks summer for %d student(s)\n", N);

for(i = 0; i < N; i++)
{
        printf("-------------------------------------------------------\n");
        printf("Enter marks for Student#%d\n", i+1);
        printf("Please provide marks for five subjects\n");
        printf("Enter the marks in the format obtained/maximum\n");
        printf("Example:\n");
        printf("90/100\n");
        total_marks[i] = total_maximum_marks[i] = 0;
        for(j = 0; j < 5; j++)
        {
                scanf("%d/%d", &marks[i][j], &max[i][j]);
                total_marks[i] += marks[i][j];
                total_maximum_marks[i] += max[i][j];
        }
}

printf("Here are the sum of marks for all students:\n");

for(i = 0; i < N; i++)
        printf("Student#%d: %d/%d\n", i+1, total_marks[i], total_maximum_marks[i]);
```

In particular, see the use of a single loop vs two loops with nesting

Convince yourself that these two code fragments are equivalent

Homework !!

The mechanism of storing values at addresses that we discussed is called the *Row-major* method

- Arrays in C are stored in row-major format
- There is another mechanism for storing arrays, called the *Column-major* method
- Read more about Column-major method, and try to think when it may be preferrable over row-major method