Introduction to Programming

Week – *3*, Lecture – *2*

Arrays in C – Part 1

SAURABH SRIVASTAVA

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

IIT KANPUR

The Marks Calculator !!

Let us make the lives of our user a little easier

The Marks Calculator !!

Let us make the lives of our user a little easier

Why not sum up the marks and maximum marks for them as well

The Marks Calculator !!

```c
#include<stdio.h>

int main()
{
        int total_marks, total_maximum_marks;
        int marks1, marks2, marks3, marks4, marks5;
        int max1, max2, max3, max4, max5;

        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");

        scanf("%d/%d", &marks1, &max1);
        scanf("%d/%d", &marks2, &max2);
        scanf("%d/%d", &marks3, &max3);
        scanf("%d/%d", &marks4, &max4);
        scanf("%d/%d", &marks5, &max5);

        total_marks = marks1 + marks2 + marks3 + marks4 + marks5;
        total_maximum_marks = max1 + max2 + max3 + max4 + max5;

        printf("Total obtained marks: %d\n", total_marks);

        printf("Total maximum marks: %d\n", total_maximum_marks);

        return 0;
}
```

The Marks Calculator !!

```c
#include<stdio.h>

int main()
{
        int total_marks, total_maximum_marks;
        int marks1, marks2, marks3, marks4, marks5;
        int max1, max2, max3, max4, max5;

        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");

        scanf("%d/%d", &marks1, &max1);
        scanf("%d/%d", &marks2, &max2);
        scanf("%d/%d", &marks3, &max3);
        scanf("%d/%d", &marks4, &max4);
        scanf("%d/%d", &marks5, &max5);

        total_marks = marks1 + marks2 + marks3 + marks4 + marks5;
        total_maximum_marks = max1 + max2 + max3 + max4 + max5;

        printf("Total obtained marks: %d\n", total_marks);

        printf("Total maximum marks: %d\n", total_maximum_marks);

        return 0;
}
```

To store obtained and maximum marks for 5 subjects, we need 10 `int` variables

The Marks Calculator !!

Let us make the lives of our user a little easier

Why not sum up the marks and maximum marks for them as well

Now imagine, if we had more subjects !!

The Marks Calculator !!

Let us make the lives of our user a little easier

Why not sum up the marks and maximum marks for them as well

Now imagine, if we had more subjects !!

Declaring so many variables, with all of them doing the same job (storing marks) seems tedious

The Marks Calculator !!

Let us make the lives of our user a little easier

Why not sum up the marks and maximum marks for them as well

Now imagine, if we had more subjects !!

Declaring so many variables, with all of them doing the same job (storing marks) seems tedious

Fortunately, most programming languages have a mechanism to ease your life in such cases

The Marks Calculator !!

Let us make the lives of our user a little easier

Why not sum up the marks and maximum marks for them as well

Now imagine, if we had more subjects !!

Declaring so many variables, with all of them doing the same job (storing marks) seems tedious

Fortunately, most programming languages have a mechanism to ease your life in such cases

They provide a special type of data type, called an *Arrray*

Arrays to the rescue…

An array is an *ordered collection of individual variables*

Arrays to the rescue…

An array is an *ordered collection of individual variables*

You can access a particular variable in the collection, with its *index*

- Indices start from `0`, and go up to `size - 1`, where `size` represents the size of the array

Arrays to the rescue…

An array is an *ordered collection of individual variables*

You can access a particular variable in the collection, with its *index*

- Indices start from `0`, and go up to `size - 1`, where `size` represents the size of the array

The size of an array, is the *number of variables it contains*

- The size of the array must be provided explicitly by the programmer
- Also, once provided, it *cannot be changed*; it remains the same throughout the program

Arrays to the rescue…

An array is an *ordered collection of individual variables*

You can access a particular variable in the collection, with its *index*

- Indices start from `0`, and go up to `size - 1`, where `size` represents the size of the array

The size of an array, is the *number of variables it contains*

- The size of the array must be provided explicitly by the programmer
- Also, once provided, it *cannot be changed*; it remains the same throughout the program

To create an array, you need to supply two pieces of information

- The type of variables this array will contain
- The number of variables that this array will contain

Creating Arrays in C

We create an array in C by providing its type and its size

- Something like:
  ```
  int arr[10];
  ```
- The general syntax is `<type of array> <name of array>[<size of array];`

Creating Arrays in C

We create an array in C by providing its type and its size

- Something like:
  `int arr[10];`
- The general syntax is `<type of array> <name of array>[<size of array];`

In earlier versions of C, the size *must* be a constant, however recent versions allow variables too

- So, while in current versions, the following statements are legal:
  `int size = 10; int arr[size];`
- … it is better to keep the size of the array a constant, to keep it compatible over all platforms

Creating Arrays in C

We create an array in C by providing its type and its size

- Something like:
  `int arr[10];`
- The general syntax is `<type of array> <name of array>[<size of array];`

In earlier versions of C, the size *must* be a constant, however recent versions allow variables too

- So, while in current versions, the following statements are legal:
  `int size = 10; int arr[size];`
- … it is better to keep the size of the array a constant, to keep it compatible over all platforms

Remember, even if you use a variable as the size, the array itself remains the same size

- Even if the value of the used variable is changed later

Creating Arrays in C

We create an array in C by providing its type and its size

- Something like:
  `int arr[10];`
- The general syntax is `<type of array> <name of array>[<size of array];`

In earlier versions of C, the size *must* be a constant, however recent versions allow variables too

- So, while in current versions, the following statements are legal:
  `int size = 10; int arr[size];`
- … it is better to keep the size of the array a constant, to keep it compatible over all platforms

Remember, even if you use a variable as the size, the array itself remains the same size

- Even if the value of the used variable is changed later

You can also *initialise* the variables at different indices in an array, at the time of creation like:

```
int arr[5] = {10, 20, 30, 40 ,50};
```

The Marks Calculator returns !!

```c
#include<stdio.h>

int main()
{
        int total_marks, total_maximum_marks;
        int marks[5];
        int max[5];

        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");

        scanf("%d/%d", &marks[0], &max[0]);
        scanf("%d/%d", &marks[1], &max[1]);
        scanf("%d/%d", &marks[2], &max[2]);
        scanf("%d/%d", &marks[3], &max[3]);
        scanf("%d/%d", &marks[4], &max[4]);

        total_marks = marks[0] + marks[1] + marks[2] + marks[3] + marks[4];
        total_maximum_marks = max[0] + max[1] + max[2] + max[3] + max[4];

        printf("Total obtained marks: %d\n", total_marks);

        printf("Total maximum marks: %d\n", total_maximum_marks);

        return 0;
}
```

The Marks Calculator returns !!

```c
#include<stdio.h>

int main()
{
        int total_marks, total_maximum_marks;
        int marks[5];
        int max[5];

        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");

        scanf("%d/%d", &marks[0], &max[0]);
        scanf("%d/%d", &marks[1], &max[1]);
        scanf("%d/%d", &marks[2], &max[2]);
        scanf("%d/%d", &marks[3], &max[3]);
        scanf("%d/%d", &marks[4], &max[4]);

        total_marks = marks[0] + marks[1] + marks[2] + marks[3] + marks[4];
        total_maximum_marks = max[0] + max[1] + max[2] + max[3] + max[4];

        printf("Total obtained marks: %d\n", total_marks);

        printf("Total maximum marks: %d\n", total_maximum_marks);

        return 0;
}
```

We are now using two arrays to store the required information

The Marks Calculator returns !!

```c
#include<stdio.h>

int main()
{
        int total_marks, total_maximum_marks;
        int marks[5];
        int max[5];

        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");

        scanf("%d/%d", &marks[0], &max[0]);
        scanf("%d/%d", &marks[1], &max[1]);
        scanf("%d/%d", &marks[2], &max[2]);
        scanf("%d/%d", &marks[3], &max[3]);
        scanf("%d/%d", &marks[4], &max[4]);

        total_marks = marks[0] + marks[1] + marks[2] + marks[3] + marks[4];
        total_maximum_marks = max[0] + max[1] + max[2] + max[3] + max[4];

        printf("Total obtained marks: %d\n", total_marks);

        printf("Total maximum marks: %d\n", total_maximum_marks);

        return 0;
}
```

… and that's how we access individual variables, by putting their index inside square brackets

Arrays and Strings in C

Strings are just a special type of array

Arrays and Strings in C

Strings are just a special type of array

They are `char` arrays, with a special character to mark their end

- This character is called the *null character*, and is denoted by `\0`
- Remember the “escape characters” that we discussed w.r.t. `printf()` – `\0` is another example

Arrays and Strings in C

Strings are just a special type of array

They are `char` arrays, with a special character to mark their end

- This character is called the *null character*, and is denoted by `\0`
- Remember the "escape characters" that we discussed w.r.t. `printf()` – `\0` is another example

In addition to usual initialisation mechanism, C supports another way to initialise char arrays

Arrays and Strings in C

Strings are just a special type of array

They are `char` arrays, with a special character to mark their end

- This character is called the *null character*, and is denoted by `\0`
- Remember the "escape characters" that we discussed w.r.t. `printf()` – `\0` is another example

In addition to usual initialisation mechanism, C supports another way to initialise char arrays

- So, both these are legal in C:

```
char str[5] = {'H', 'i', '\0'};
char str[5] = "Hi";
```

Arrays and Strings in C

Strings are just a special type of array

They are `char` arrays, with a special character to mark their end

- This character is called the *null character*, and is denoted by `\0`
- Remember the "escape characters" that we discussed w.r.t. `printf()` – `\0` is another example

In addition to usual initialisation mechanism, C supports another way to initialise char arrays

- So, both these are legal in C:

```
char str[5] = {'H', 'i', '\0'};
char str[5] = "Hi";
```

- In the latter case, we don't need to provide the null character explicitly – it is added implicitly

Arrays and Strings in C

Strings are just a special type of array

They are `char` arrays, with a special character to mark their end

- This character is called the *null character*, and is denoted by `\0`
- Remember the "escape characters" that we discussed w.r.t. `printf()` – `\0` is another example

In addition to usual initialisation mechanism, C supports another way to initialise char arrays

- So, both these are legal in C:

```
char str[5] = {'H', 'i', '\0'};
char str[5] = "Hi";
```

- In the latter case, we don't need to provide the null character explicitly – it is added implicitly
- Note that in both cases, we are only using 3 out of the 5 character slots in the array

Arrays and Strings in C

Strings are just a special type of array

They are `char` arrays, with a special character to mark their end

- This character is called the *null character*, and is denoted by `\0`
- Remember the "escape characters" that we discussed w.r.t. `printf()` – `\0` is another example

In addition to usual initialisation mechanism, C supports another way to initialise char arrays

- So, both these are legal in C:

```
char str[5] = {'H', 'i', '\0'};
char str[5] = "Hi";
```

- In the latter case, we don't need to provide the null character explicitly – it is added implicitly
- Note that in both cases, we are only using 3 out of the 5 character slots in the array
- … the last two slots will have some unpredictable characters – we also call such values as *garbage values*

Arrays and Strings in C

Strings are just a special type of array

They are `char` arrays, with a special character to mark their end

- This character is called the *null character*, and is denoted by `\0`
- Remember the "escape characters" that we discussed w.r.t. `printf()` – `\0` is another example

In addition to usual initialisation mechanism, C supports another way to initialise char arrays

- So, both these are legal in C:

```
char str[5] = {'H', 'i', '\0'};
char str[5] = "Hi";
```

- In the latter case, we don't need to provide the null character explicitly – it is added implicitly
- Note that in both cases, we are only using 3 out of the 5 character slots in the array
- … the last two slots will have some unpredictable characters – we also call such values as *garbage values*

We will spend more time specifically on arrays towards the end of the course

A simple string…

```c
#include<stdio.h>

int main()
{
        char string[5] = "Hi";
        // This is what gets stored in string:
        // Index ->  0  |  1  |  2  |  3  |  4
        // char  -> 'H' | 'i' |'\0' |  X  |  X
        // X implies some "garbage value"!!
        // i.e., some value that we don't care about...
        // By the way, what you are reading right now,
        // are called "comments"...
        // The compiler "ignores" any content on the current line,
        // that is preceded by two slashes, i.e. //

        puts(string);

        printf("Tell me a 4-letter word: ");
        gets(string);

        printf("You just said - ");
        puts(string);
}
~
~
~
~
~
```

A simple string…

```c
#include<stdio.h>

int main()
{
        char string[5] = "Hi";
        // This is what gets stored in string:
        // Index ->  0  |  1  |  2  |  3  |  4
        // char  -> 'H' | 'i' |'\0' |  X  |  X
        // X implies some "garbage value"!!
        // i.e., some value that we don't care about...
        // By the way, what you are reading right now,
        // are called "comments"...
        // The compiler "ignores" any content on the current line,
        // that is preceded by two slashes, i.e. //

        puts(string);

        printf("Tell me a 4-letter word: ");
        gets(string);

        printf("You just said - ");
        puts(string);
}
~
~
~
~
~
```

You should use a character array of ”sufficient” size to store a string, including the null character

A simple string…

```c
#include<stdio.h>

int main()
{
        char string[5] = "Hi";
        // This is what gets stored in string:
        // Index ->  0  |  1  |  2  |  3  |  4
        // char  -> 'H' | 'i' |'\0' |  X  |  X
        // X implies some "garbage value"!!
        // i.e., some value that we don't care about...
        // By the way, what you are reading right now,
        // are called "comments"...
        // The compiler "ignores" any content on the current line,
        // that is preceded by two slashes, i.e. //

        puts(string);

        printf("Tell me a 4-letter word: ");
        gets(string);

        printf("You just said - ");
        puts(string);
}
~
~
~
~
~
```

Two library functions that can take string inputs and show strings as outputs are `gets()` and `puts()`

A simple string…

```
saurabh@saurabh-VirtualBox:~/C/examples/Week 3$ vim Stringify.c
saurabh@saurabh-VirtualBox:~/C/examples/Week 3$ gcc -o Stringify String
ify.c
Stringify.c: In function ‘main’:
Stringify.c:19:2: warning: implicit declaration of function ‘gets’; did
 you mean ‘fgets’? [-Wimplicit-function-declaration]
  gets(string);
  ^~~~
  fgets
/tmp/cc8LHIpw.o: In function `main':
Stringify.c:(.text+0x4c): warning: the `gets' function is dangerous and
 should not be used.
saurabh@saurabh-VirtualBox:~/C/examples/Week 3$ ./Stringify
Hi
Tell me a 4-letter word: cool
You just said - cool
saurabh@saurabh-VirtualBox:~/C/examples/Week 3$
```

Check out your Homework !!

Homework !!

There are many interesting ways in which arrays can be initialised, read more about them

- This link has a nice picture: https://www.geeksforgeeks.org/arrays-in-c-cpp/

There is a format specifier, %s, which can be used with `scanf()` for taking string inputs

- There is, however, a behaviour of `scanf()`, which restricts the kind of strings it can take as inputs
- Find out what is that behaviour !!

The last slide had a compilation warning about the `gets()` function – it says it is "dangerous"

- Find out why it is dangerous !!
- Try to change the `Stringify.c` in such a way that it produces the same output, but there are no warnings