Introduction to Programming

Week – *3*, Lecture – *3*

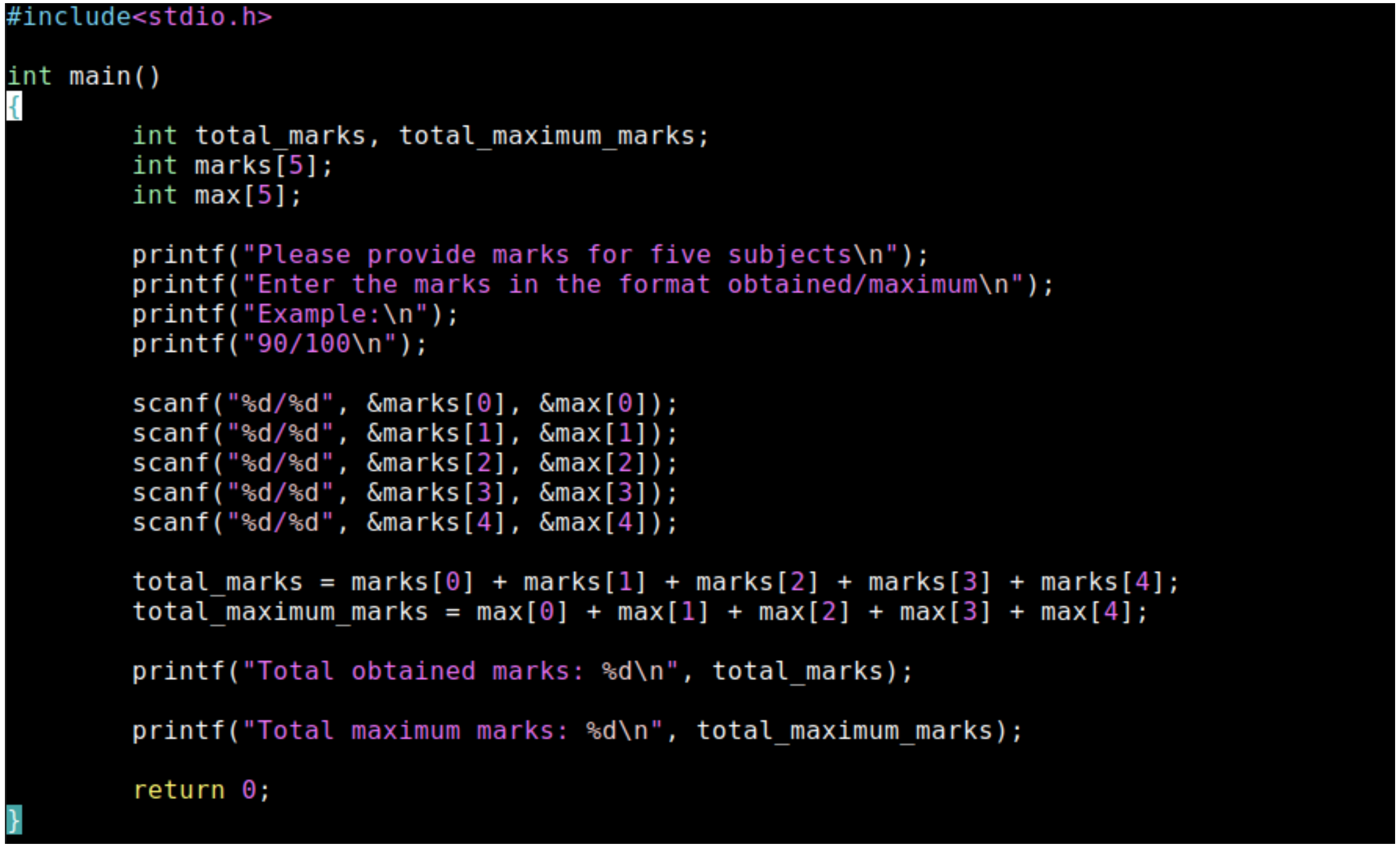
Introduction to Iterative Programming

SAURABH SRIVASTAVA

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

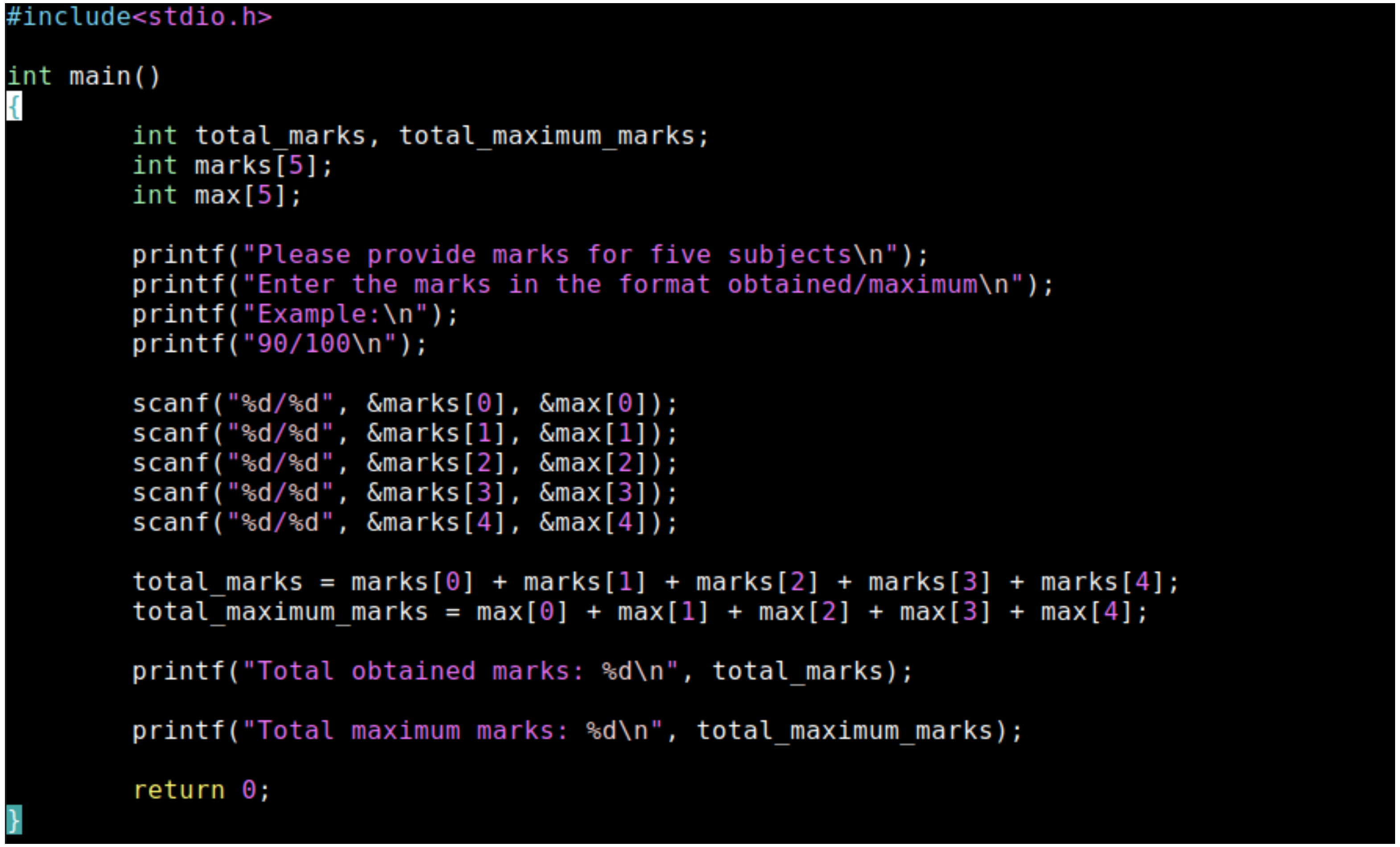
IIT KANPUR

Revisiting the Marks Summer



```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;
}
```

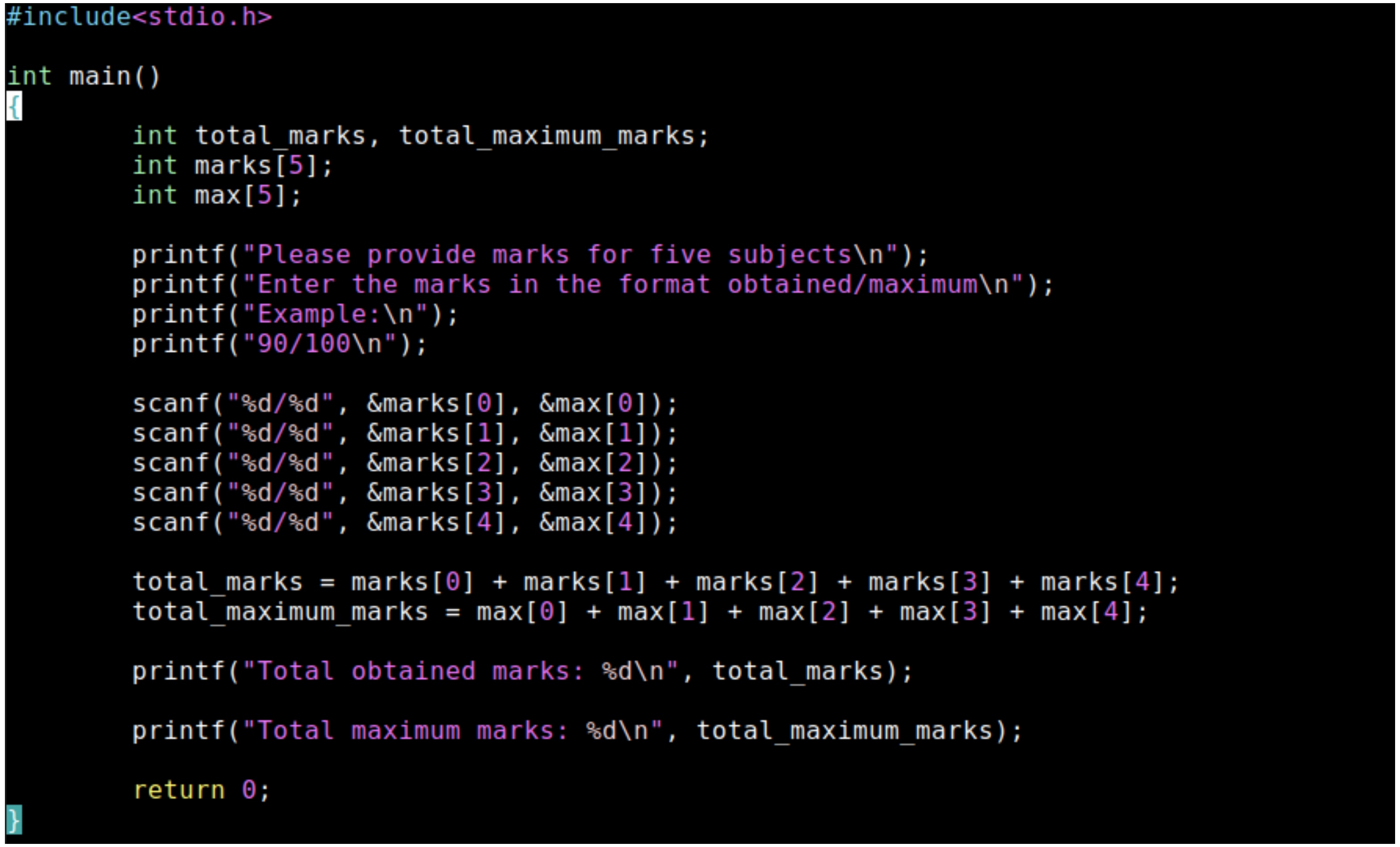
Revisiting the Marks Summer



```
#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;
}
```

While we reduced the effort required to create variables by using arrays

Revisiting the Marks Summer

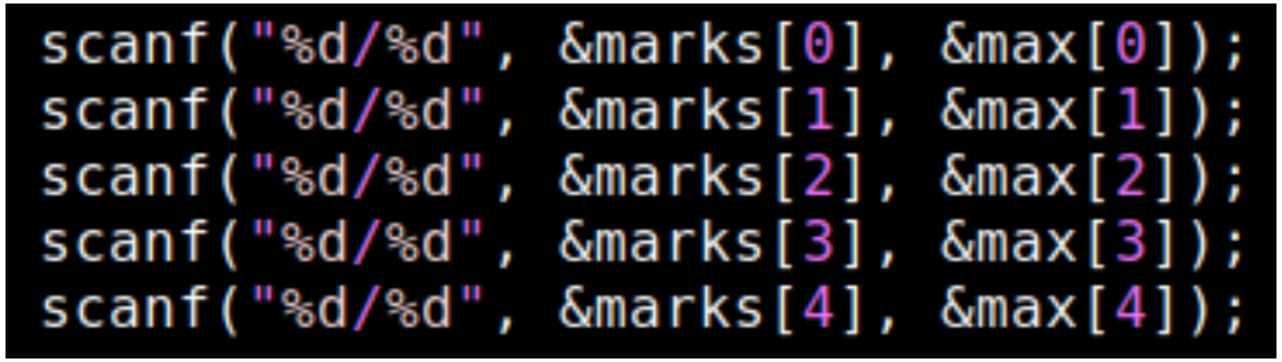


```
#include<stdio.h>

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;
}
```

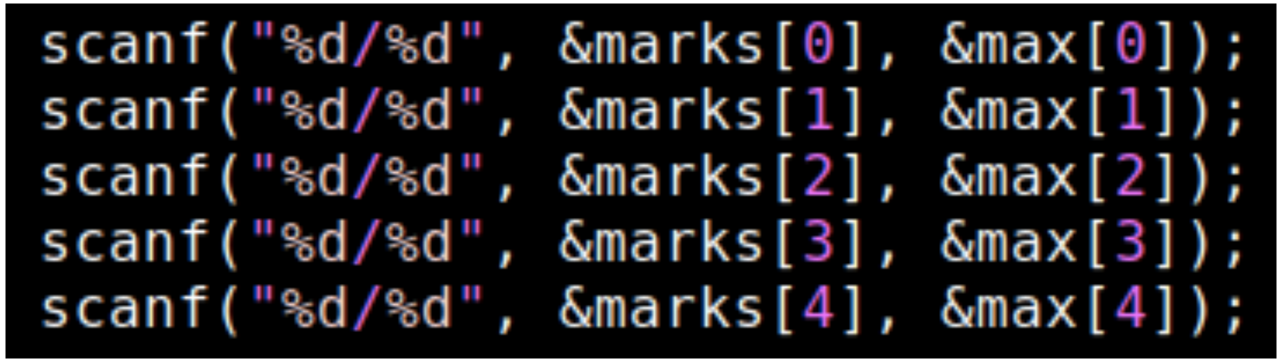
While we reduced the effort required to create variables by using arrays

We have not really reduced the total number of lines in the code !!

Running "similar" code repeatedly


```
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]);
```

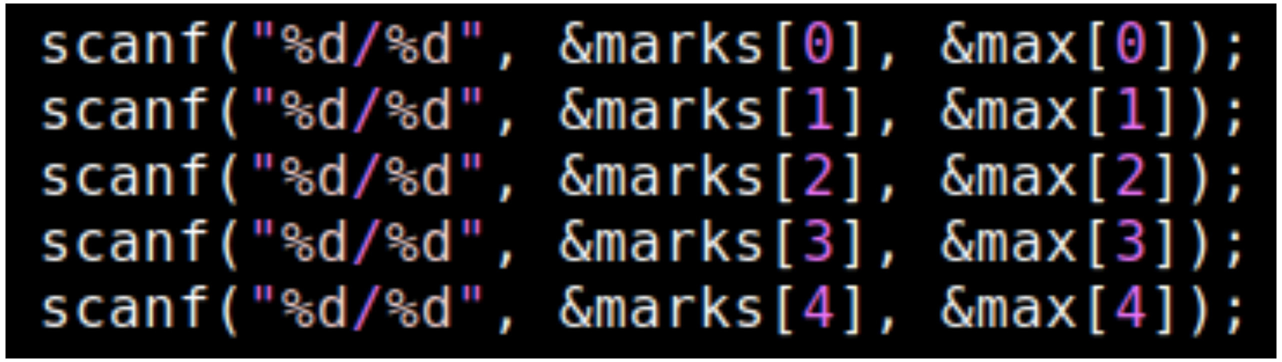
Running “similar” code repeatedly



```
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]);
```

Focus on this part of the code

Running “similar” code repeatedly

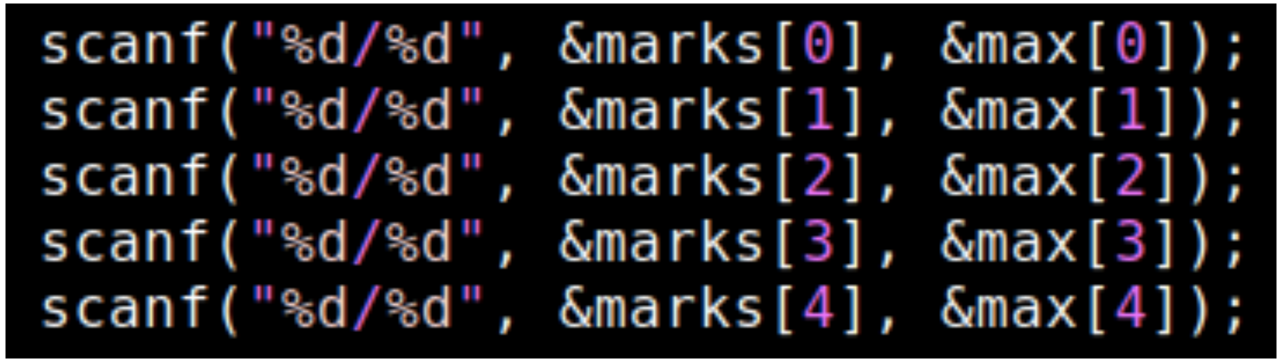


```
scanf("%d/%d", &marks[0], &max[0]);
scanf("%d/%d", &marks[1], &max[1]);
scanf("%d/%d", &marks[2], &max[2]);
scanf("%d/%d", &marks[3], &max[3]);
scanf("%d/%d", &marks[4], &max[4]);
```

Focus on this part of the code

What exactly is changing in these lines?

Running “similar” code repeatedly



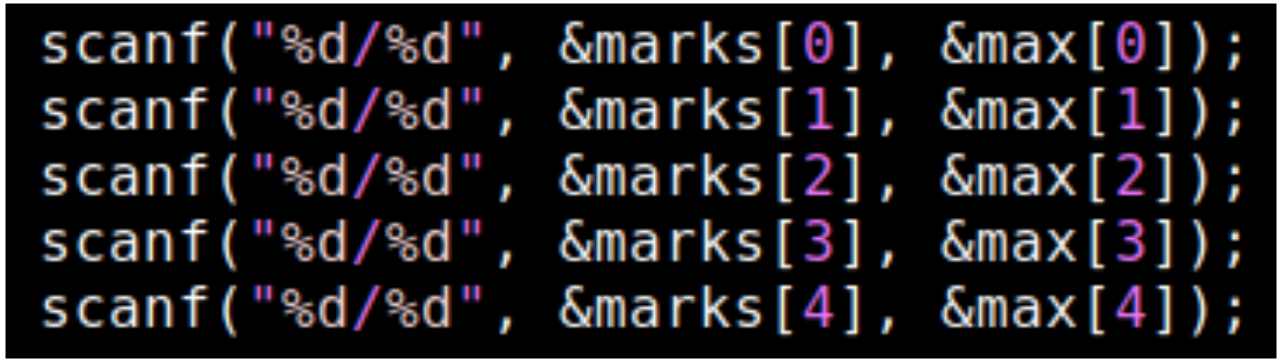
```
scanf("%d/%d", &marks[0], &max[0]);
scanf("%d/%d", &marks[1], &max[1]);
scanf("%d/%d", &marks[2], &max[2]);
scanf("%d/%d", &marks[3], &max[3]);
scanf("%d/%d", &marks[4], &max[4]);
```

Focus on this part of the code

What exactly is changing in these lines?

Just the index of the array elements !!

Running “similar” code repeatedly



```
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]);
```

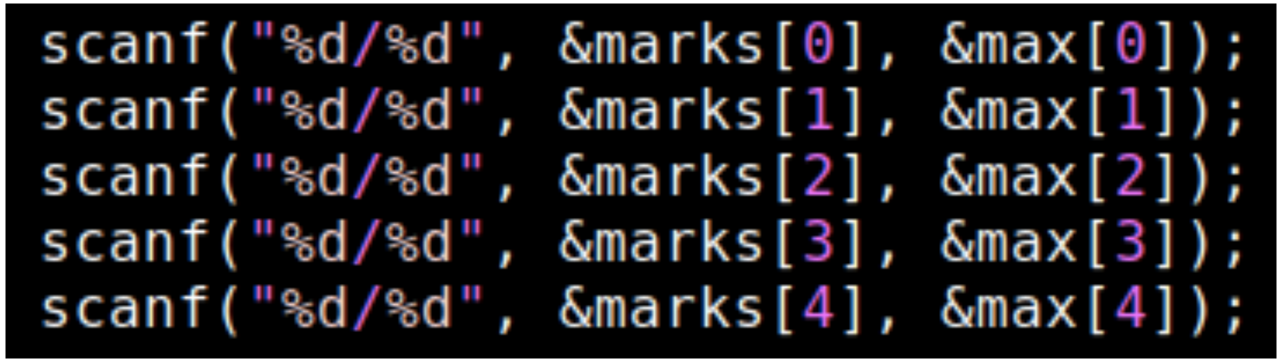
Focus on this part of the code

What exactly is changing in these lines?

Just the index of the array elements !!

What if we use a variable here for representing the index of array elements, and increase the value of that variable by 1 every time?

Running "similar" code repeatedly

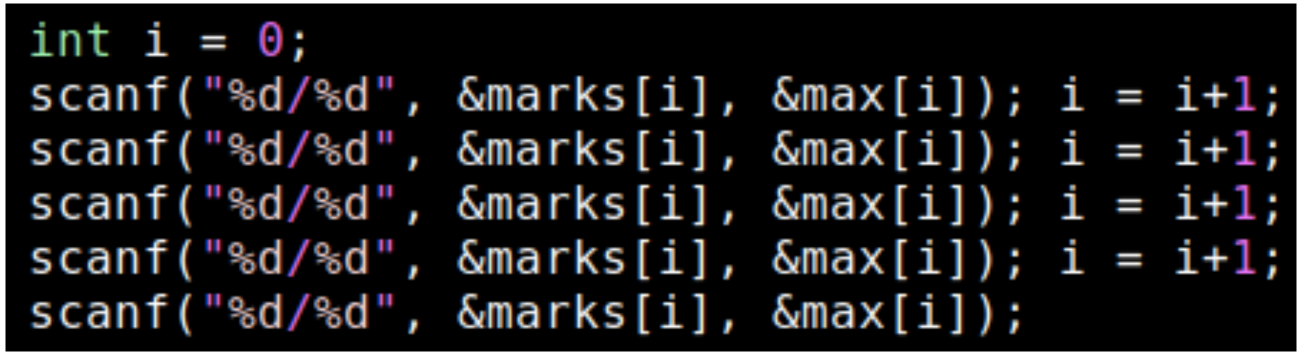


```
scanf("%d/%d", &marks[0], &max[0]);
scanf("%d/%d", &marks[1], &max[1]);
scanf("%d/%d", &marks[2], &max[2]);
scanf("%d/%d", &marks[3], &max[3]);
scanf("%d/%d", &marks[4], &max[4]);
```

Focus on this part of the code

What exactly is changing in these lines?

Just the index of the array elements !!

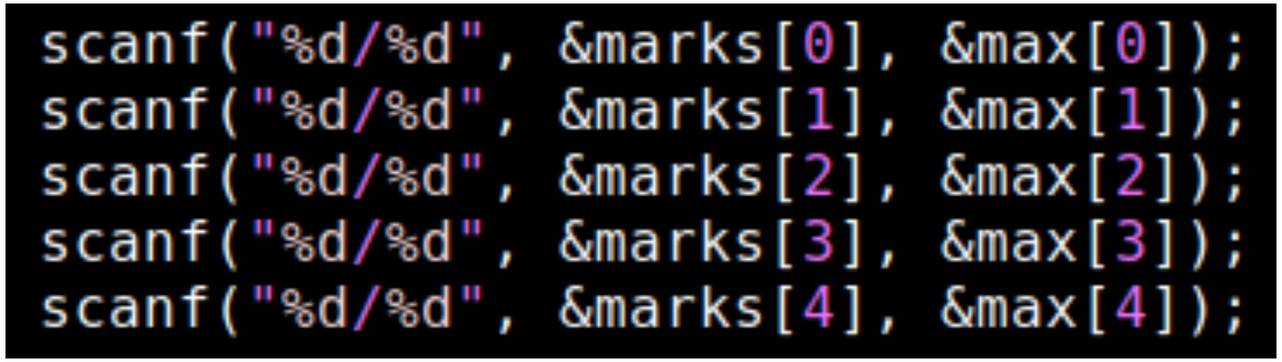


```
int i = 0;
scanf("%d/%d", &marks[i], &max[i]); i = i+1;
scanf("%d/%d", &marks[i], &max[i]); i = i+1;
scanf("%d/%d", &marks[i], &max[i]); i = i+1;
scanf("%d/%d", &marks[i], &max[i]); i = i+1;
scanf("%d/%d", &marks[i], &max[i]);
```

What if we use a variable here for representing the index of array elements, and increase the value of that variable by 1 every time?

Something like this !!

Running “similar” code repeatedly

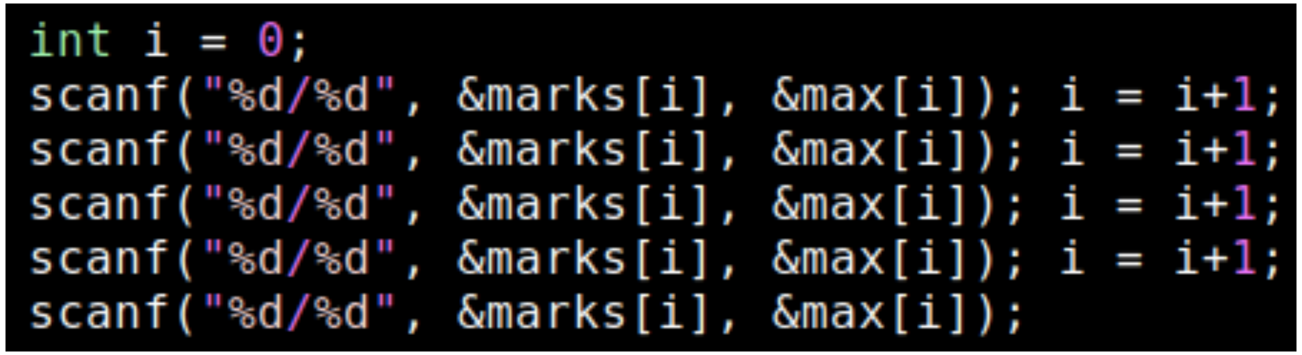


```
scanf("%d/%d", &marks[0], &max[0]);
scanf("%d/%d", &marks[1], &max[1]);
scanf("%d/%d", &marks[2], &max[2]);
scanf("%d/%d", &marks[3], &max[3]);
scanf("%d/%d", &marks[4], &max[4]);
```

Focus on this part of the code

What exactly is changing in these lines?

Just the index of the array elements !!



```
int i = 0;
scanf("%d/%d", &marks[i], &max[i]); i = i+1;
scanf("%d/%d", &marks[i], &max[i]); i = i+1;
scanf("%d/%d", &marks[i], &max[i]); i = i+1;
scanf("%d/%d", &marks[i], &max[i]); i = i+1;
scanf("%d/%d", &marks[i], &max[i]);
```

What if we use a variable here for representing the index of array elements, and increase the value of that variable by 1 every time?

Something like this !!

By the way, this is a common statement in programming:

`i = i + 1;`

It means, "take the value of `i`, add `1` to it, and store it back in `i`"

In terms of algorithm, what we want is…

```
i = 0
X:
    Take inputs in marks[i] and max[i]
    increase the value of i by 1
    if i <= 4, go to X
```

In terms of algorithm, what we want is…

```
i = 0
X:
    Take inputs in marks[i] and max[i]
    increase the value of i by 1
    if i <= 4, go to X
```

Here, ***X*** is a label

In terms of algorithm, what we want is…

```
i = 0
X:
    Take inputs in marks[i] and max[i]
    increase the value of i by 1
    if i <= 4, go to X
```

Here, ***X*** is a label

Labels denote a position in the algorithm

In terms of algorithm, what we want is...

```
i = 0
X:
    Take inputs in marks[i] and max[i]
    increase the value of i by 1
    if i <= 4, go to X
```

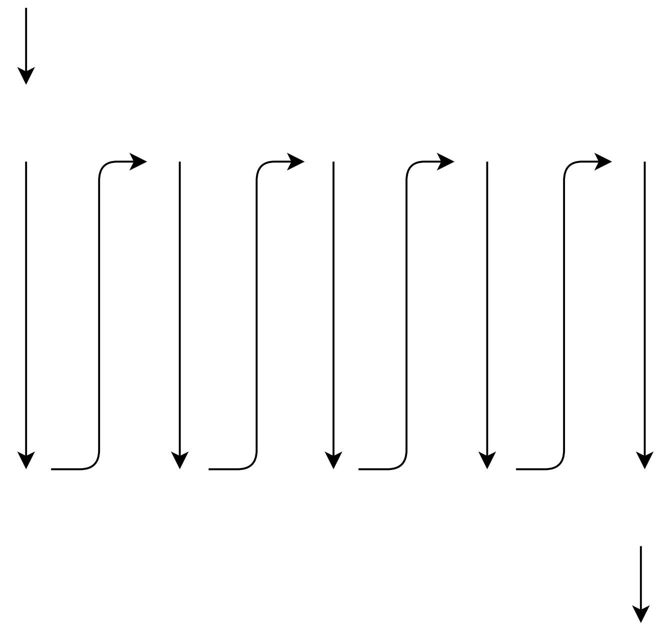
Here, ***X*** is a label

Labels denote a position in the algorithm

The "go to" instruction is a *branching* instruction here

In terms of algorithm, what we want is…

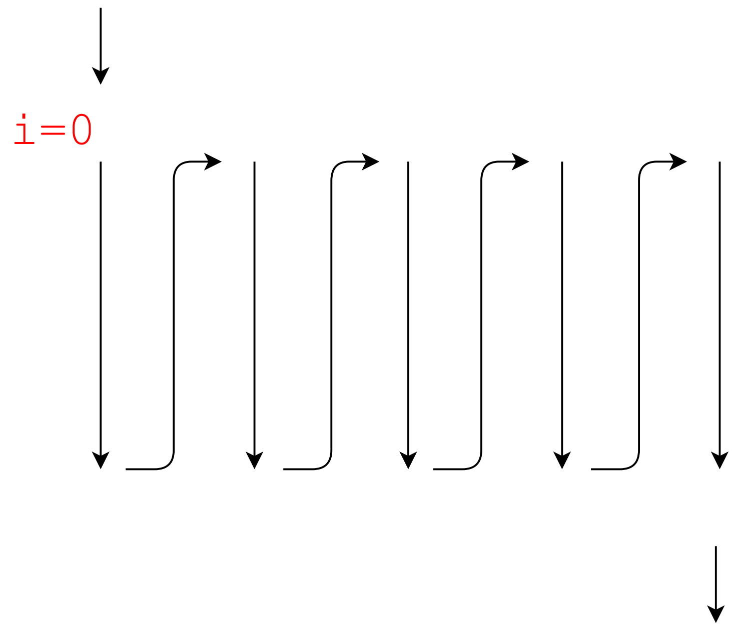
```
i = 0
X:
    Take inputs in marks[i] and max[i]
    increase the value of i by 1
    if i <= 4, go to X
```



This is how the instructions here are executed

In terms of algorithm, what we want is...

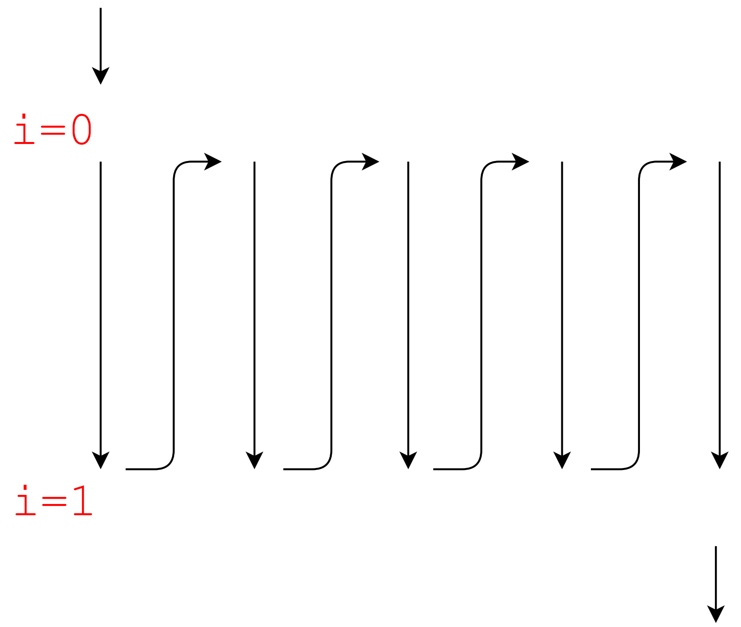
```
i = 0
X:
    Take inputs in marks[i] and max[i]
    increase the value of i by 1
    if i <= 4, go to X
```



This is how the instructions here are executed

In terms of algorithm, what we want is…

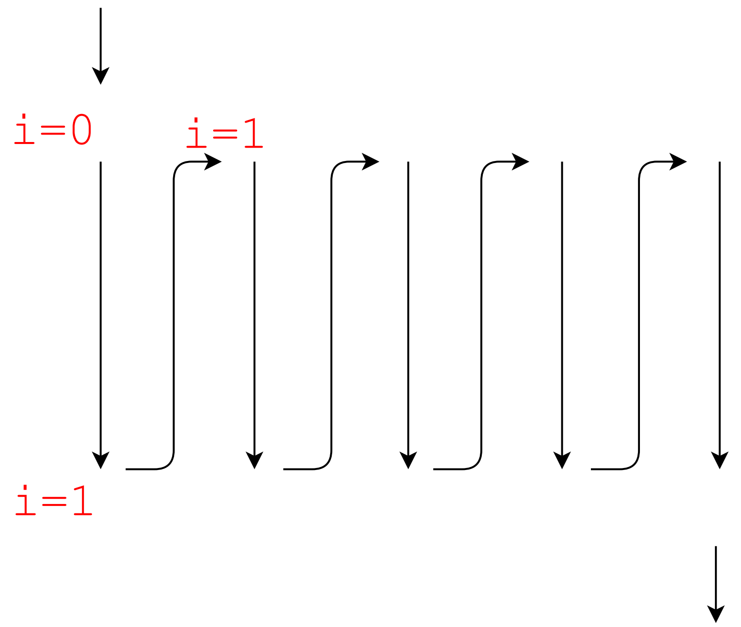
```
i = 0
X:
    Take inputs in marks[i] and max[i]
    increase the value of i by 1
    if i <= 4, go to X
```



This is how the instructions here are executed

In terms of algorithm, what we want is…

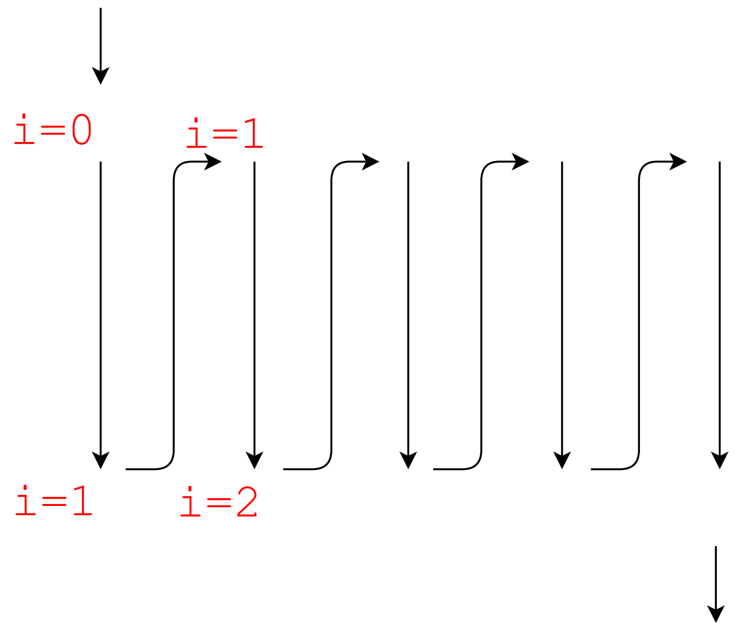
```
i = 0
X:
    Take inputs in marks[i] and max[i]
    increase the value of i by 1
    if i <= 4, go to X
```



This is how the instructions here are executed

In terms of algorithm, what we want is…

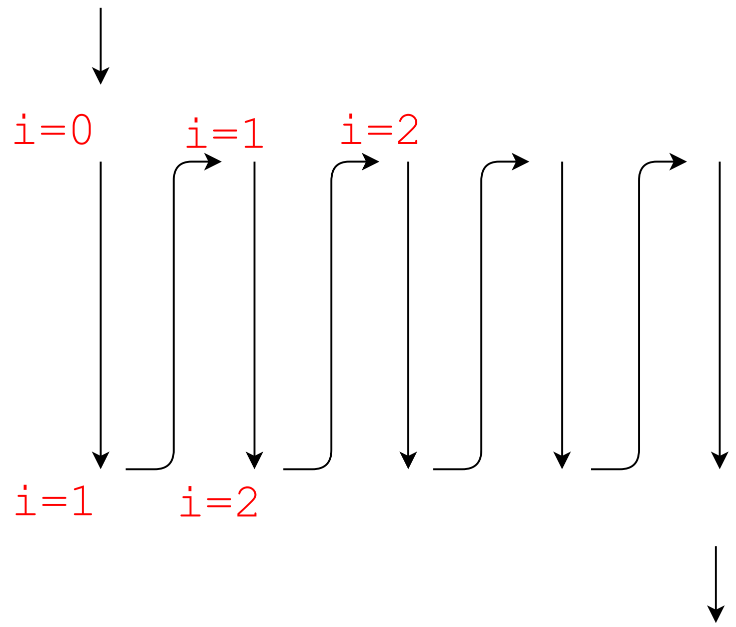
```
i = 0
X:
    Take inputs in marks[i] and max[i]
    increase the value of i by 1
    if i <= 4, go to X
```



This is how the instructions here are executed

In terms of algorithm, what we want is…

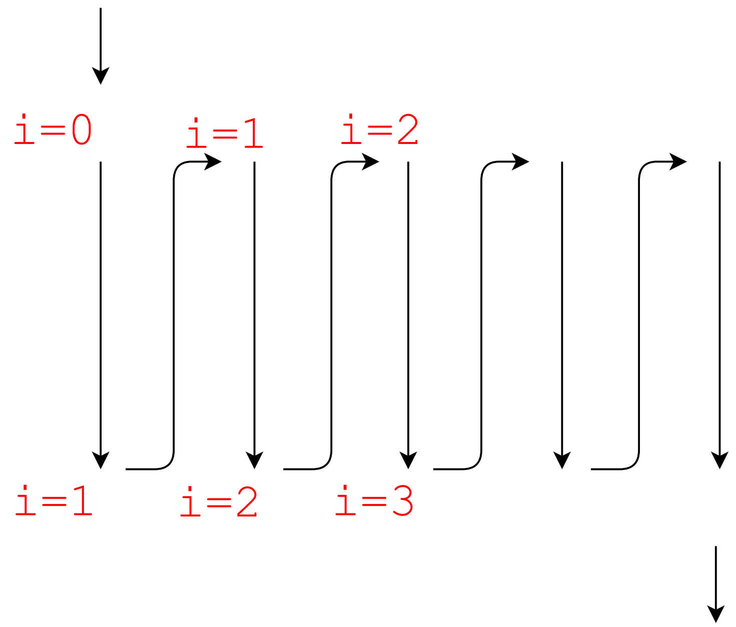
```
i = 0
X:
    Take inputs in marks[i] and max[i]
    increase the value of i by 1
    if i <= 4, go to X
```



This is how the instructions here are executed

In terms of algorithm, what we want is…

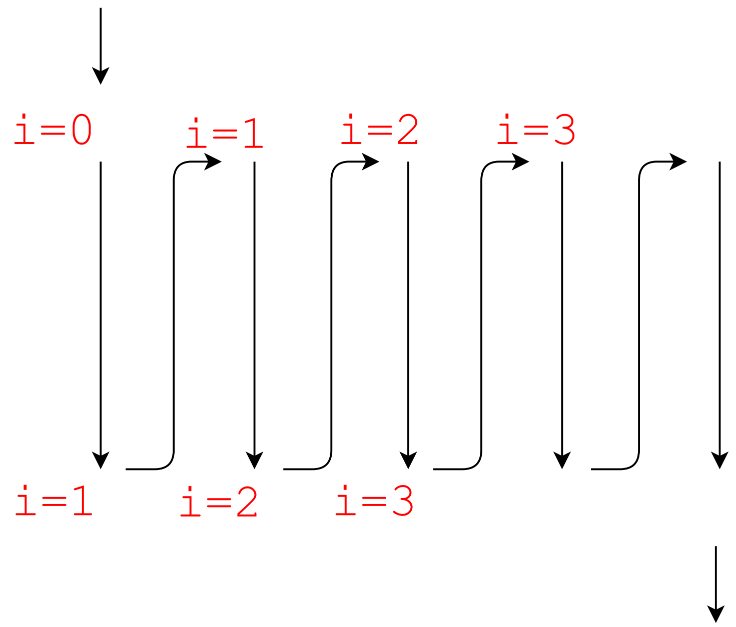
```
i = 0
X:
    Take inputs in marks[i] and max[i]
    increase the value of i by 1
    if i <= 4, go to X
```



This is how the instructions here are executed

In terms of algorithm, what we want is…

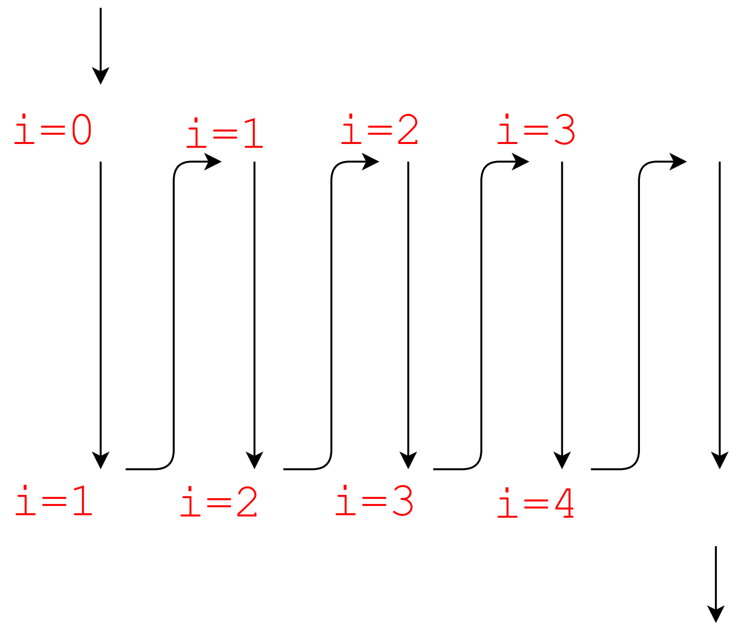
```
i = 0
X:
    Take inputs in marks[i] and max[i]
    increase the value of i by 1
    if i <= 4, go to X
```



This is how the instructions here are executed

In terms of algorithm, what we want is…

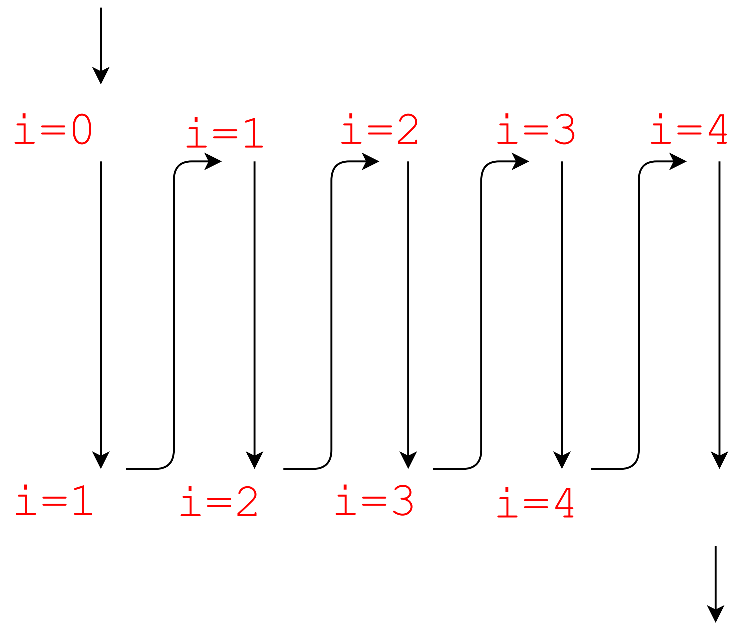
```
i = 0
X:
    Take inputs in marks[i] and max[i]
    increase the value of i by 1
    if i <= 4, go to X
```



This is how the instructions here are executed

In terms of algorithm, what we want is…

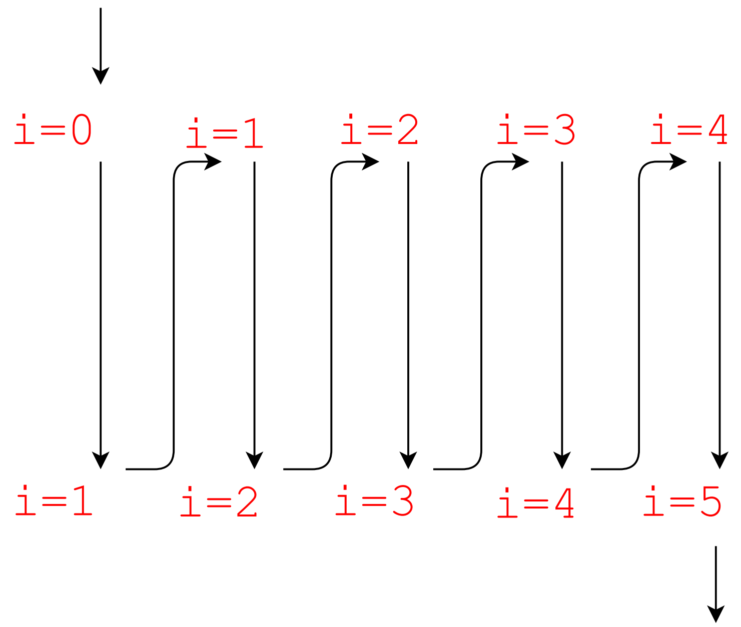
```
i = 0
X:
    Take inputs in marks[i] and max[i]
    increase the value of i by 1
    if i <= 4, go to X
```



This is how the instructions here are executed

In terms of algorithm, what we want is…

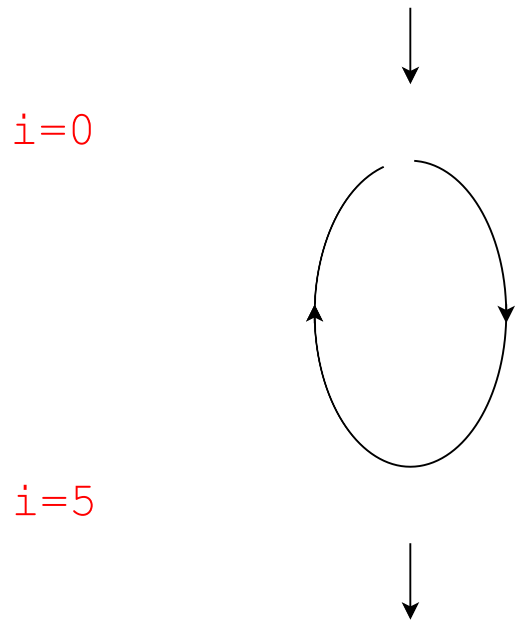
```
i = 0
X:
    Take inputs in marks[i] and max[i]
    increase the value of i by 1
    if i <= 4, go to X
```



At this point the instruction right after the last statement above, is executed

In terms of algorithm, what we want is…

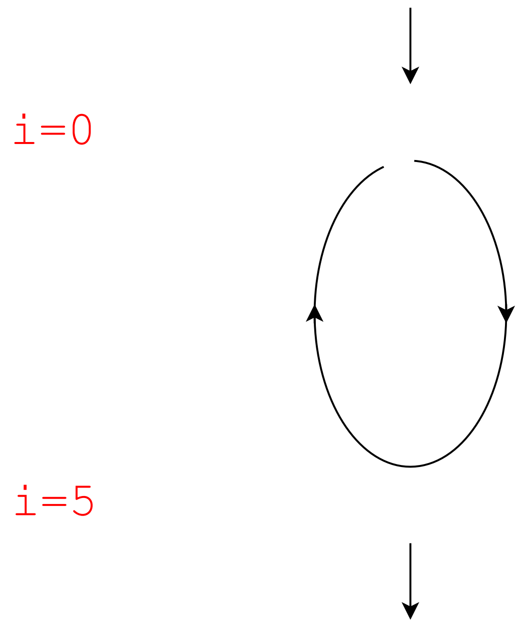
```
i = 0
X:
    Take inputs in marks[i] and max[i]
    increase the value of i by 1
    if i <= 4, go to X
```



Another way to think about this flow is as shown

In terms of algorithm, what we want is…

```
i = 0
X:
    Take inputs in marks[i] and max[i]
    increase the value of i by 1
    if i <= 4, go to X
```



Another way to think about this flow is as shown

This type of flow in a program is called a *loop*

Why do we use loops?

Loops are used when we have to perform “similar” tasks repeatedly

Why do we use loops?

Loops are used when we have to perform “similar” tasks repeatedly

You’ll soon realise that the power of a program lies in its ability to do repeated tasks efficiently

- For example, it can do a task repeatedly, millions or billions of time !!

Why do we use loops?

Loops are used when we have to perform “similar” tasks repeatedly

You’ll soon realise that the power of a program lies in its ability to do repeated tasks efficiently

- For example, it can do a task repeatedly, millions or billions of time !!

You can solve a number of real-world problems by writing one or more loops

- Actually, this repeated processing is far more common than you may expect !!

Why do we use loops?

Loops are used when we have to perform "similar" tasks repeatedly

You'll soon realise that the power of a program lies in its ability to do repeated tasks efficiently

- For example, it can do a task repeatedly, millions or billions of time !!

You can solve a number of real-world problems by writing one or more loops

- Actually, this repeated processing is far more common than you may expect !!

What you have to think about, is what changes in each *iteration* of the loop

- One iteration means the *body* of the loop being executed once
- The body, in turn, is the set of statements that are repeated multiple times

Using loops to solve problems

```
Statements before the loop
```

Using loops to solve problems

Statements before the loop

Initialisation – setting values for important variables

Using loops to solve problems

```
Statements before the loop

Initialisation – setting values for important variables

{

    Statements, using different values of one or more variables



}
```

Using loops to solve problems

```
Statements before the loop
Initialisation – setting values for important variables
{
    Statements, using different values of one or more variables
    Update to the values of one or more variables


}
```

Using loops to solve problems

```
Statements before the loop
Initialisation – setting values for important variables
{
    Statements, using different values of one or more variables
    Update to the values of one or more variables
    condition over one or more variables (to go out of the loop)
}
```

Using loops to solve problems

```
Statements before the loop
Initialisation – setting values for important variables
{
    Statements, using different values of one or more variables
    Update to the values of one or more variables
    condition over one or more variables (to go out of the loop)
}
Statements after the loop
```

Lets calculate Factorial !!

Procedure FactorialCalculater

```
    Inputs: num

    result = 1;

    if(num == 0)
    {
        return as Output : 1
    }

    do:
        result = result * num;
        num = num - 1;
        if(num > 1)
        {
            go to do;
        }

    return as Output : result
```

Lets calculate Factorial !!

Procedure FactorialCalculater

```
    Inputs: num

    result = 1;

    if(num == 0)
    {
        return as Output : 1
    }

    do:
        result = result * num;
        num = num - 1;
        if(num > 1)
        {
            go to do;
        }

    return as Output : result
```

Convince yourself that you understand this pseudocode !!

Homework !!

Consider the example from the file `Stringify.c` in Lecture *3.2*

- If you don't know which example is it, please see the `Example Programs` folder on Course's Drive Folder
- Write pseudocode for the `puts()` function, using a loop

Rewrite the Procedure `FactorialCalculator` in such a way that

- there is another variable, `i`, which is initialised to 0, and
- the statement
  ```
  num = num - 1;
  ```
- … is replaced by the statement
  ```
  i = i + 1;
  ```
- … keeping the output exactly the same