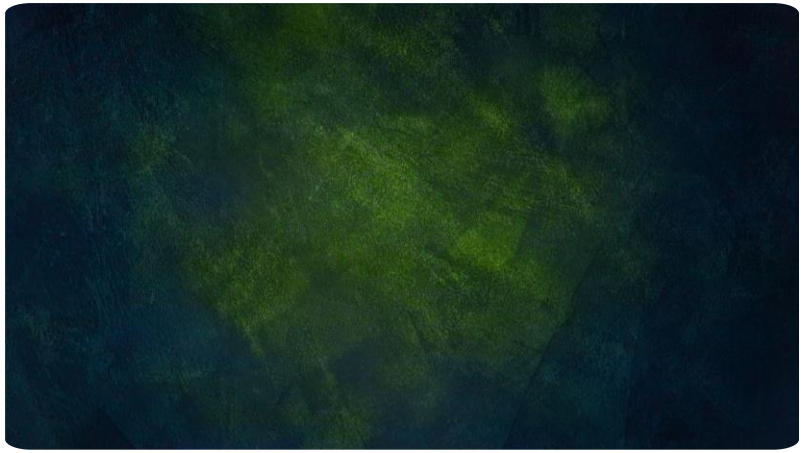
Introduction to Computing and Programming

Loops Practice & Arrays

Recap

- Loops
- Types of Loops
- Exercise on Loops
- Quiz 1 discussion, CR discussion, & LASC tutor discussion



Content

Some more Exercise on Loops

Arrays

Loop

- A loop is a sequence of instructions that is **continually repeated** until a certain condition is reached.

- They reduce the need for repetitive coding and **improve efficiency**.

- There are **three types** of loops:
  - Using a while statement
  - Using a for statement
  - Using a do-while statement

Loops Description

Loop Type	Description
While loop	Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
For loop	Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.
Do……While loop	Like a while statement, except that it tests the condition at the end of the loop body

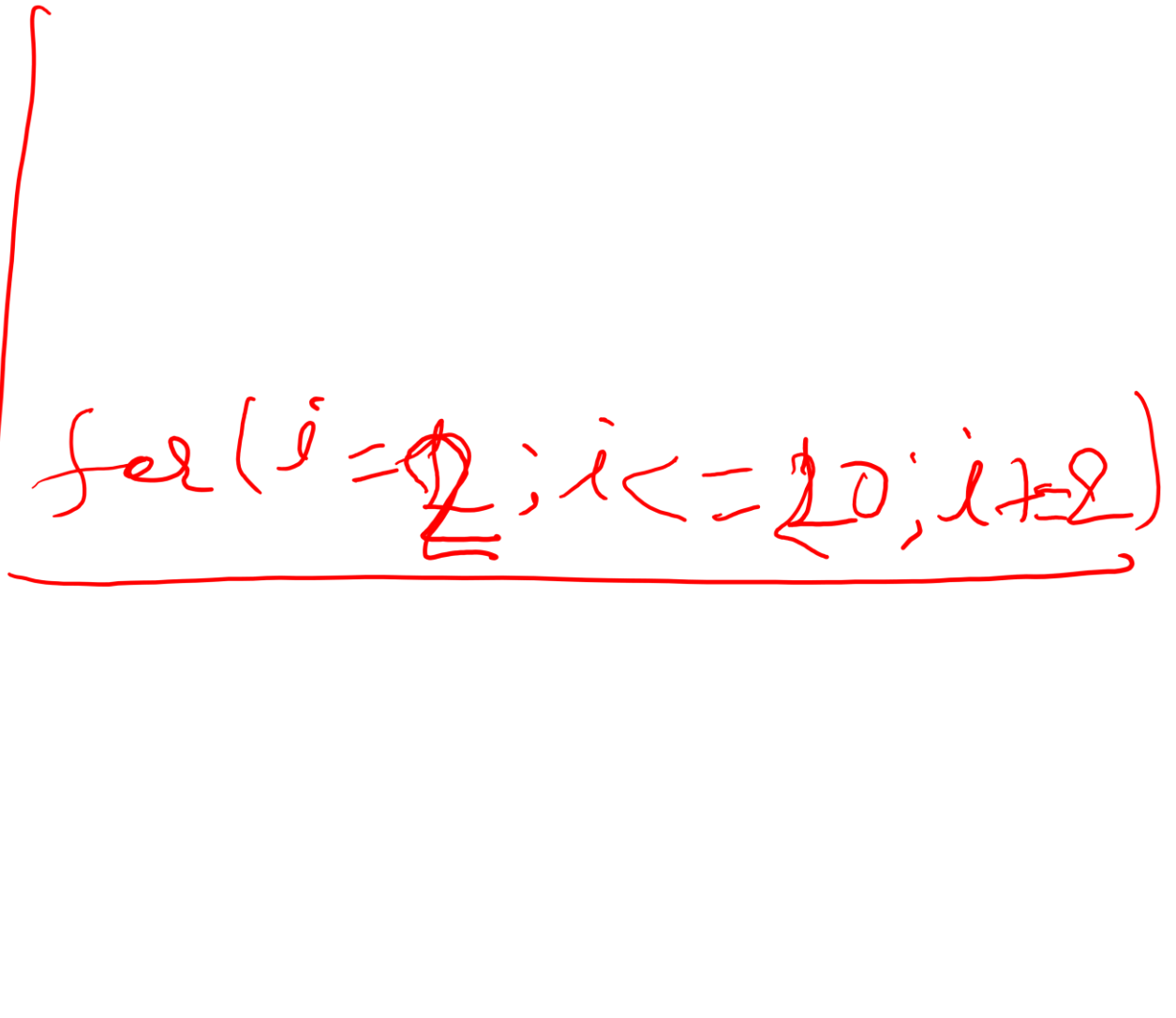
Write a C program to find sum of n natural numbers

```
#include <stdio.h>
int main() {
   int n, sum = 0;
   // Input the value of n
   printf("Enter a positive integer: ");
   scanf("%d", &n);
   // Make sure the input is a positive integer
   if (n < 0) {
      printf("Invalid input! Please enter a positive integer.\n");
      return 0;
   }
   // Calculate the sum of first n natural numbers using a loop
   for (int i = 1; i <= n; i++) {
      sum += i;
   }
   // Output the result
   printf("Sum of the first %d natural numbers is: %d\n", n, sum);

   return 0;
}
```

Sum = Sum + i ;

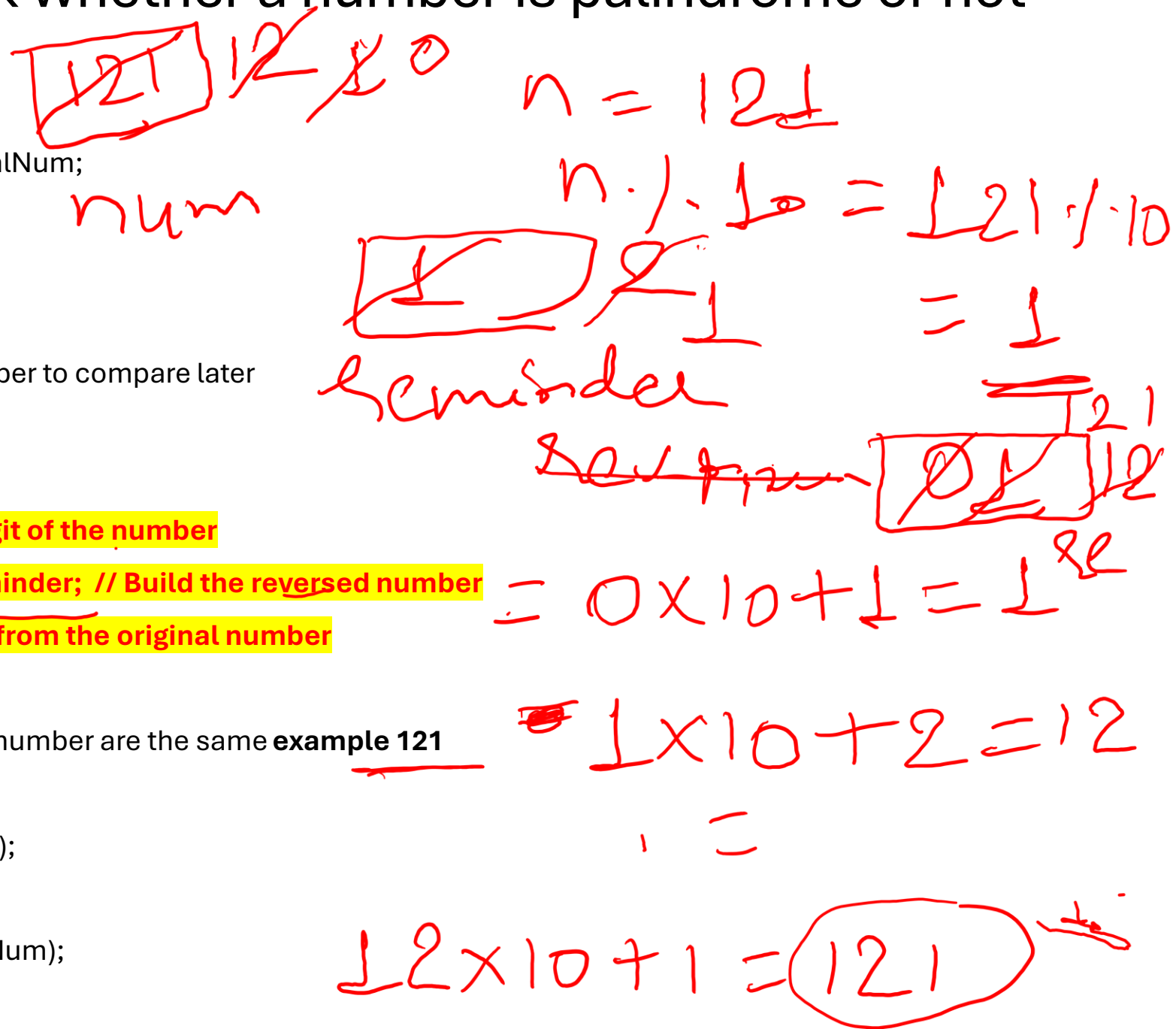
Write a C program to find the table of 2

```c
#include <stdio.h>

int main() {
    int i;

    // Print the multiplication table of 2
    printf("Multiplication table of 2:\n");
    for (i = 1; i <= 10; i++) {
        printf("2 x %d = %d\n", i, 2 * i);
    }

    return 0;
}
```



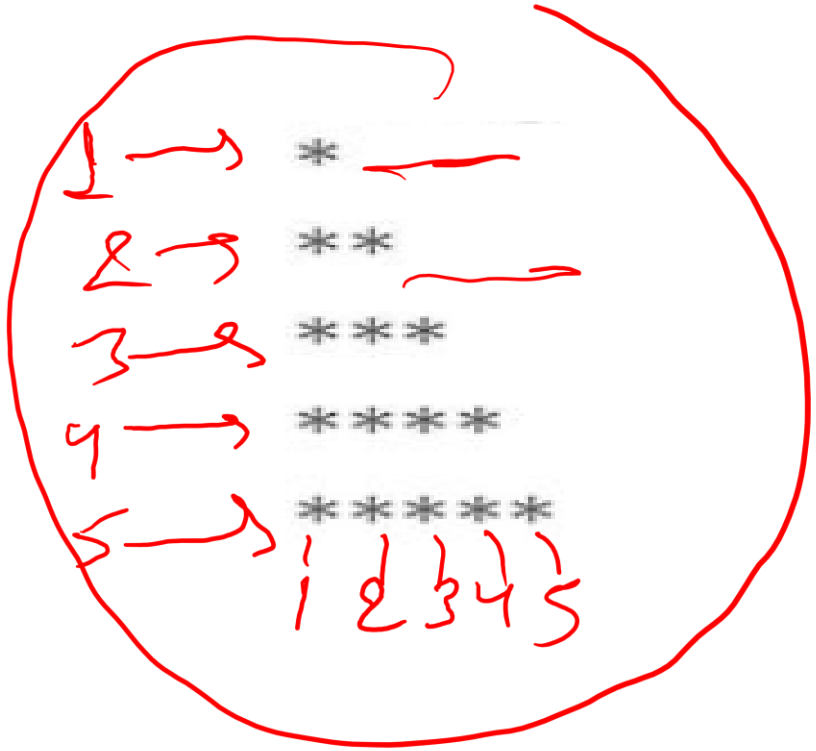
Write a C program to check whether a number is palindrome or not

```c
#include <stdio.h>
int main() {
    int num, reversedNum = 0, remainder, originalNum;
    // Input the number from the user
    printf("Enter an integer: ");
    scanf("%d", &num);
    originalNum = num; // Store the original number to compare later
    // Reverse the digits of the number
    while (num != 0) {
        remainder = num % 10;  // Get the last digit of the number
        reversedNum = reversedNum * 10 + remainder;  // Build the reversed number
        num = num / 10;  // Remove the last digit from the original number
    }
    // Check if the original number and reversed number are the same example 121
    if (originalNum == reversedNum) {
        printf("%d is a palindrome.\n", originalNum);
    } else {
        printf("%d is not a palindrome.\n", originalNum);
    } return 0; }
```



Write a C program to display a pyramid

```c
#include <stdio.h>

int main() {
    int rows = 5;  // Number of rows for the pyramid

    // Outer loop to handle the number of rows
    for (int i = 1; i <= rows; i++) {
        // Inner loop to print stars for each row
        for (int j = 1; j <= i; j++) {
            printf("*");
        }
        // Move to the next line after printing each row
        printf("\n");
    }

    return 0;
}
```



1

1 2

1 2 3

1 2 3 4

1 2 3 4 5

Write a C program to display a pyramid

```
#include <stdio.h>


int main() {
    int rows = 5;  // Number of rows for the pyramid

    // Outer loop to handle the number of rows
    for (int i = 1; i <= rows; i++) {
        // Inner loop to print stars for each row
        for (int j = 1; j <= i; j++) {
            printf("%d", j);
        }
        // Move to the next line after printing each row
        printf("\n");
    }

    return 0;
}
```

```
1
12
123
1234
12345
```

Write a C program to check whether a number is Armstrong number or not

Example:

- **153 is an Armstrong number because 1^3+5^3+3^3=153.**
- **122 is not an Armstrong number** because 1^3+2^3+2^3= 1+8+8 = 17 which is not equal to 122.
- 9474 = 9^4+4^4+7^4+4^4 is an Armstrong number.

```
#include <stdio.h>
int main() {
 int num, originalNum, remainder, result = 0;
 printf("Enter a three-digit integer: ");
 scanf("%d", &num);
 originalNum = num;
 while (originalNum != 0) {
 remainder = originalNum % 10;
 result += remainder * remainder * remainder;
 originalNum /= 10;
 }
 if (result == num)
 printf("%d is an Armstrong number.", num);
 else
 printf("%d is not an Armstrong number.", num);
 return 0; }
```

153 / 10 = 3

153 / 10 = 15

15 / 10 = 5

15 / 10 = 1

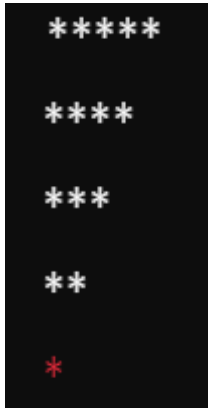
Write a C program to check whether a number is Armstrong number or not

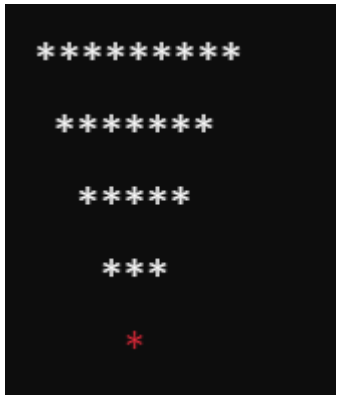
```
#include <stdio.h>
#include <math.h>
int main() {
    int num, originalNum, remainder, result = 0, n = 0;
    // Input from user
    printf("Enter an integer: ");
    scanf("%d", &num);
    originalNum = num;
    // Find the number of digits in num
    while (originalNum != 0) {
        originalNum /= 10;
        ++n;
    }
originalNum = num;
// Calculate the sum of the power of digits
    while (originalNum != 0) {
remainder = originalNum % 10;
        result += pow(remainder, n);
        originalNum /= 10;
    }

    // Check if num is an Armstrong number
    if (result == num)
        printf("%d is an Armstrong number.\n", num);
    else
        printf("%d is not an Armstrong number.\n", num);

    return 0;
}
```

Write C programs to display these patterns

https://www.geeksforgeeks.org/pattern-programs-in-c/

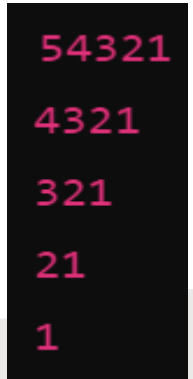
```
*****
****
***
**
*
```



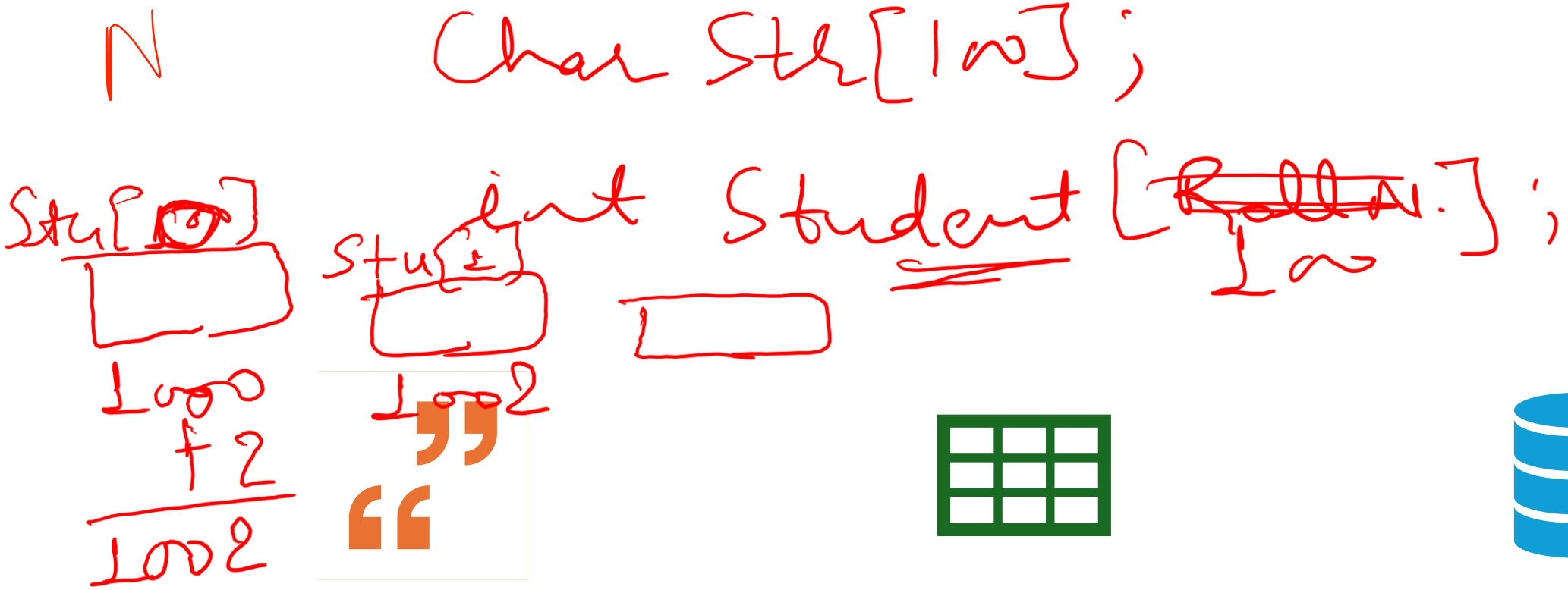


Examples of Loop for Practice

1. Write a C program to print numbers from 1 to 100
2. Write a C program to find the table of 5
3. Write a C program to check whether a number is even or not
4. Write a C program to check whether a number is Armstrong number or not
5. Write a C program to check whether a number is prime number or not
6. Write a C program to reverse the number
7. Write a C program to display Fibonacci series
8. Write a C program to print an inverted pyramid pattern



Arrays



An array is **a collection of elements of the same type** that are referenced by a common name.

Called as derived data type.

All the elements of an array occupy a set of contiguous memory locations.

Why need to use array type?

"We have a list of 1000 students' marks of an integer type. If using the basic data type (int), we will declare something like the following…“

int studMark0, studMark1, studMark2, ..., studMark999;

int studMark[1000];

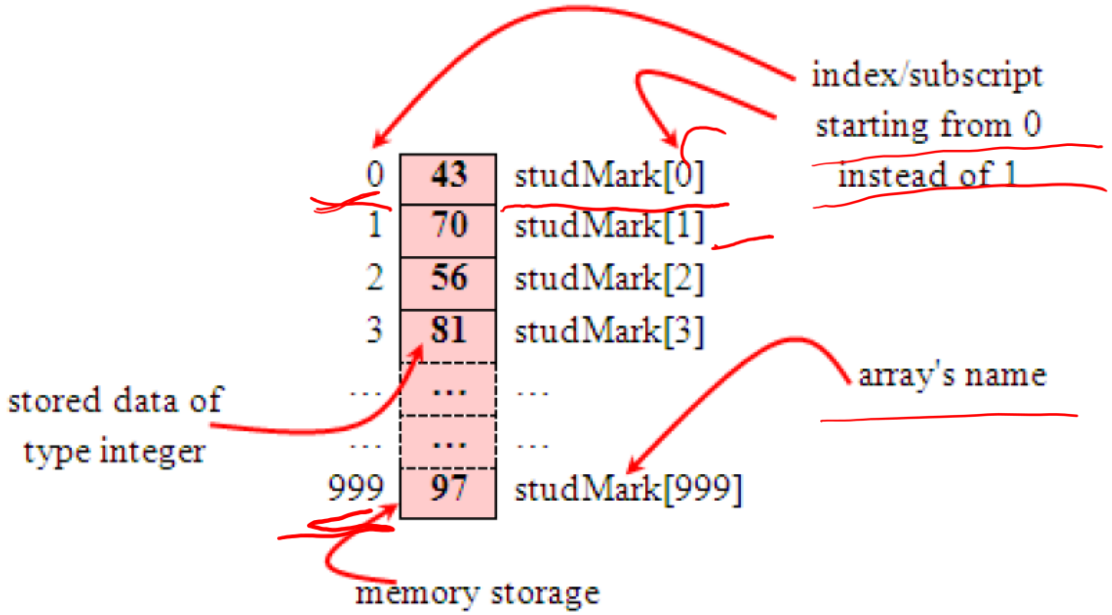
Issues if not using Arrays

- Can you imagine how long we have to write the declaration part by using normal variable declaration?

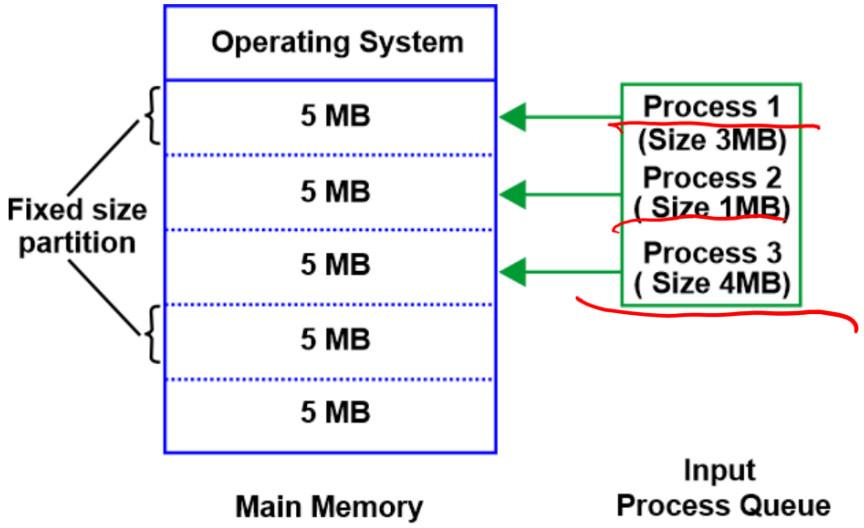
```
int main(void)
{
int studMark1, studMark2, studMark3, studMark4, …, …, studMark998,
stuMark999, studMark1000;
…
…
return 0;
}
```

Problem can be solved using Array

- By using an array, we just declare like this,
- `int studMark[1000];`
- This will reserve 1000 contiguous memory locations for storing the students' marks.
- Graphically, this can be depicted as in the following figure.

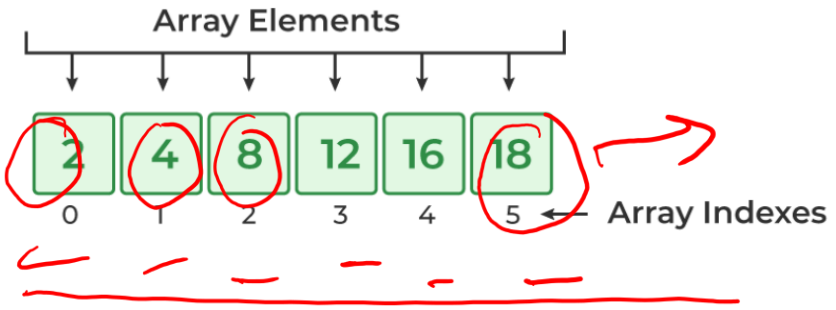


Arrays – one dimensional



- Fixed-size collection of similar data items stored in contiguous memory locations.

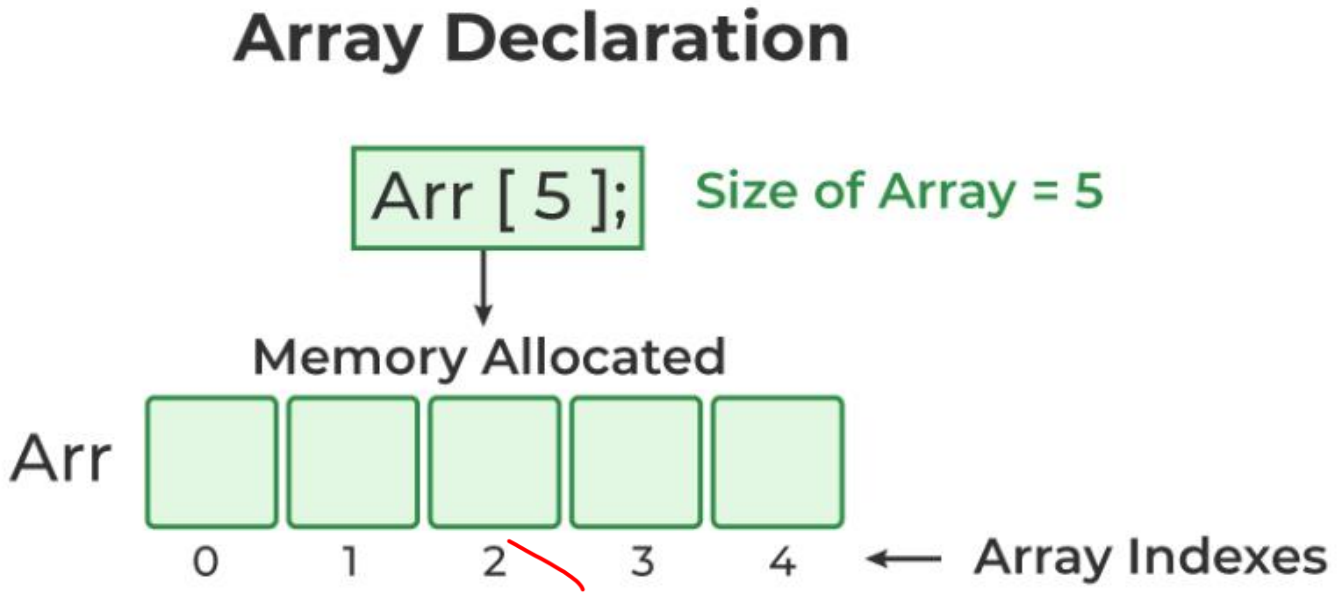
- Can be used to store the collection of primitive data types such as int, char, float, etc., as well as derived and user-defined data types such as pointers, structures, etc.



Array Declaration

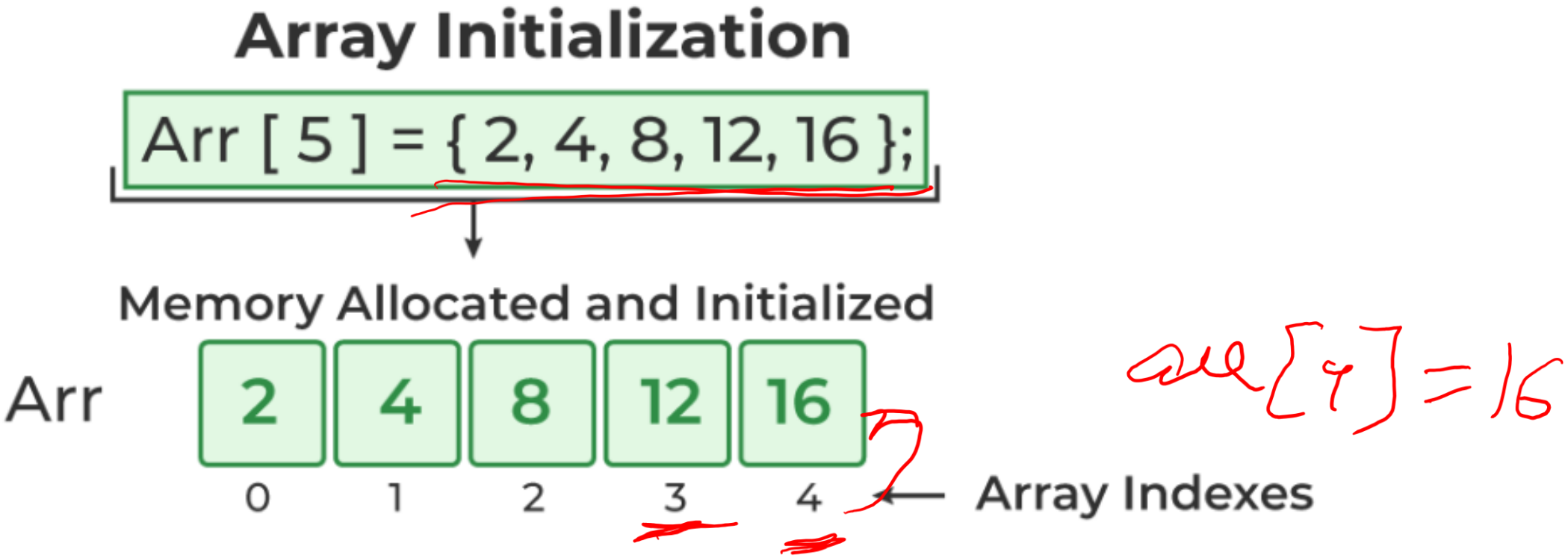
- Syntax
  - data_type array_name[array_size];
- Example:

```
int studentmark[100];
```



Array Initialization

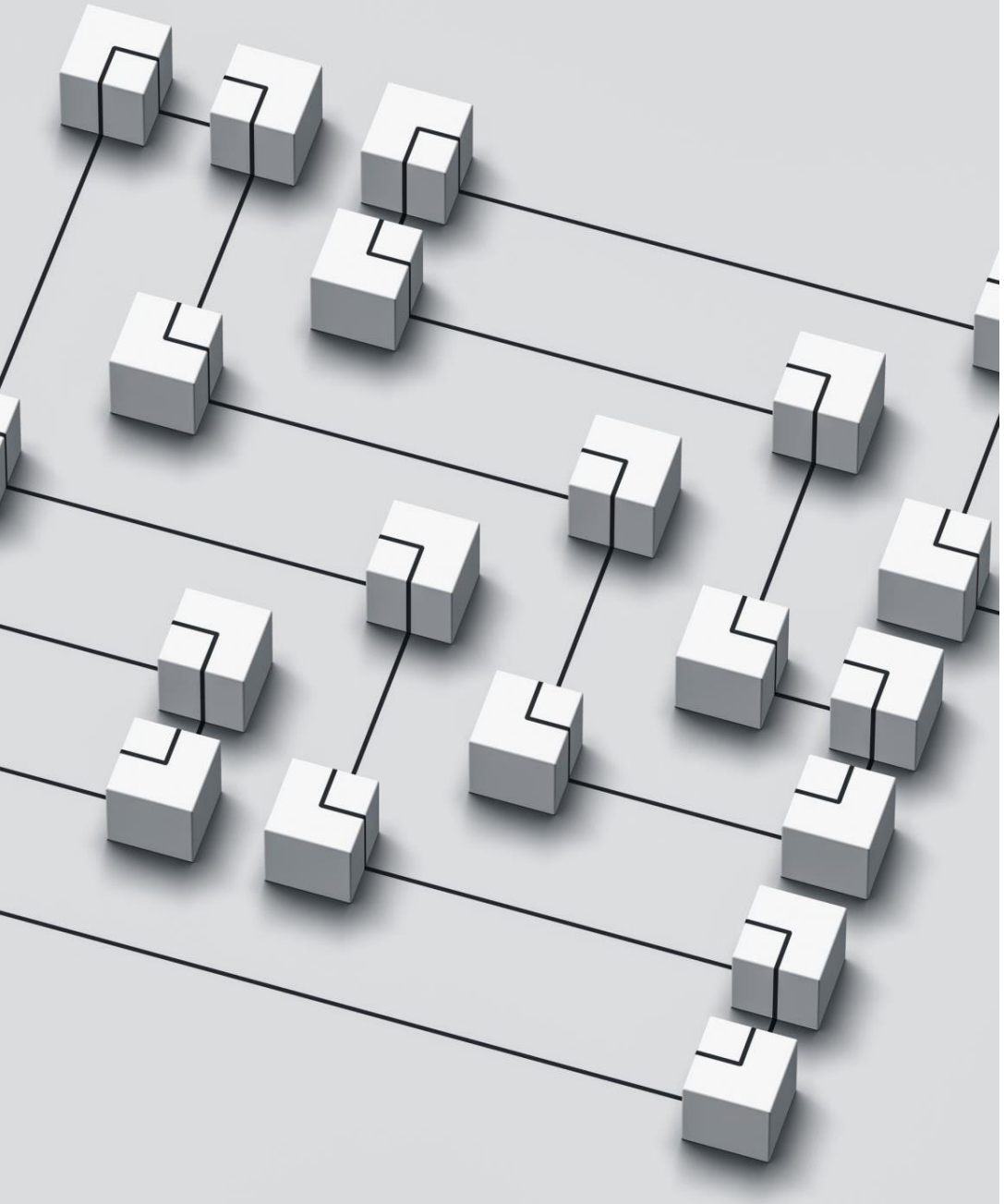
- **Initialization with Declaration**
- Syntax:
  - *data_type array_name* [*size*] = {value1, value2, ... valueN};



Array Initialization with Declaration without Size

- The compiler can automatically deduce the size of the array
- The size of the array is equal to the number of elements present in the initializer list .

- data_type array_name[] = {1,2,3,4,5};
- The size of the above arrays is 5 which is automatically deduced by the compiler.



Upcoming Slides

- Operations of Arrays
- **Some more examples of Arrays**
- **Arrays - Two dimensional**
- Array with pointer will be discussed later