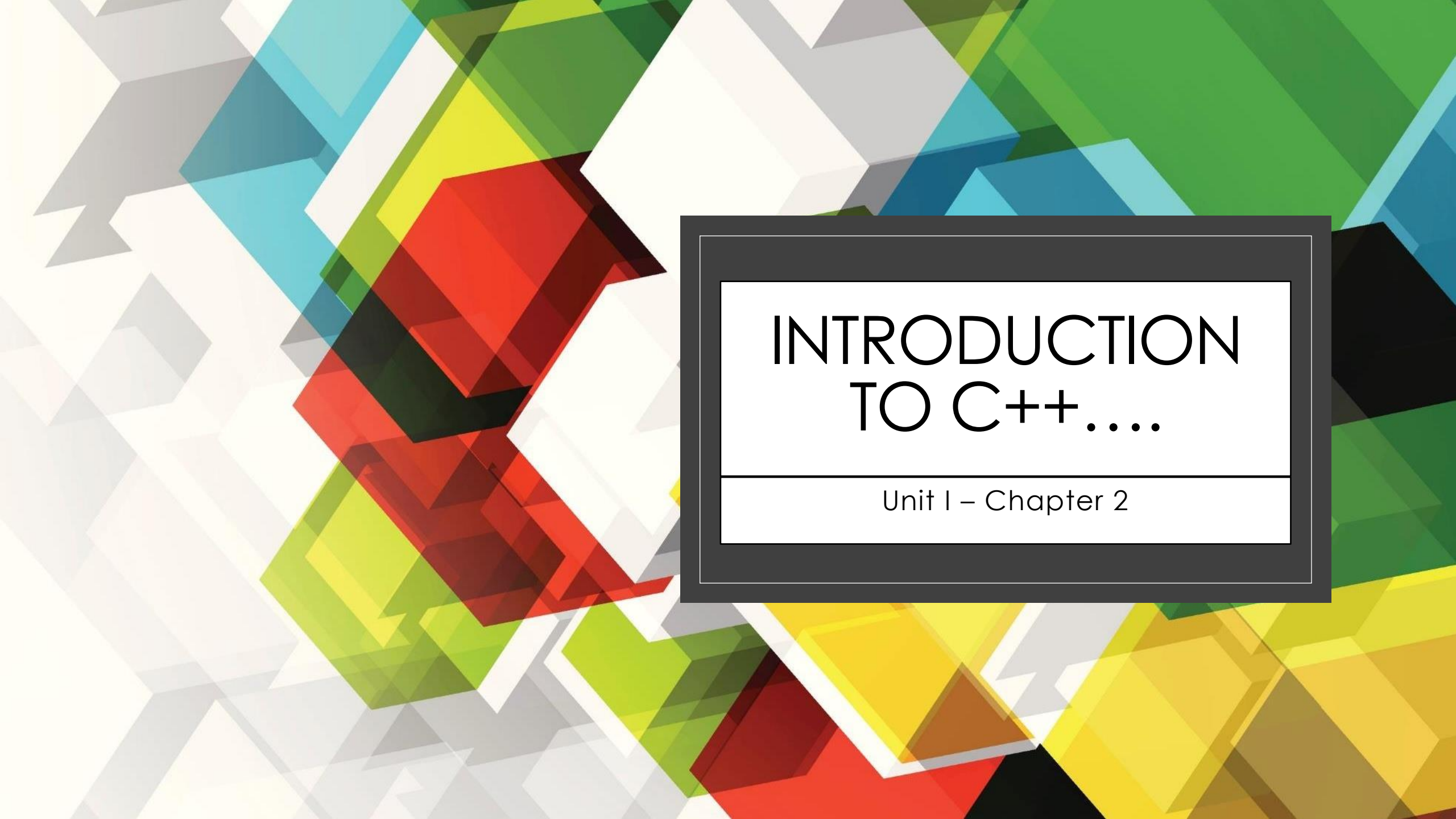

INTRODUCTION TO C++….
Unit I – Chapter 2

Arrays

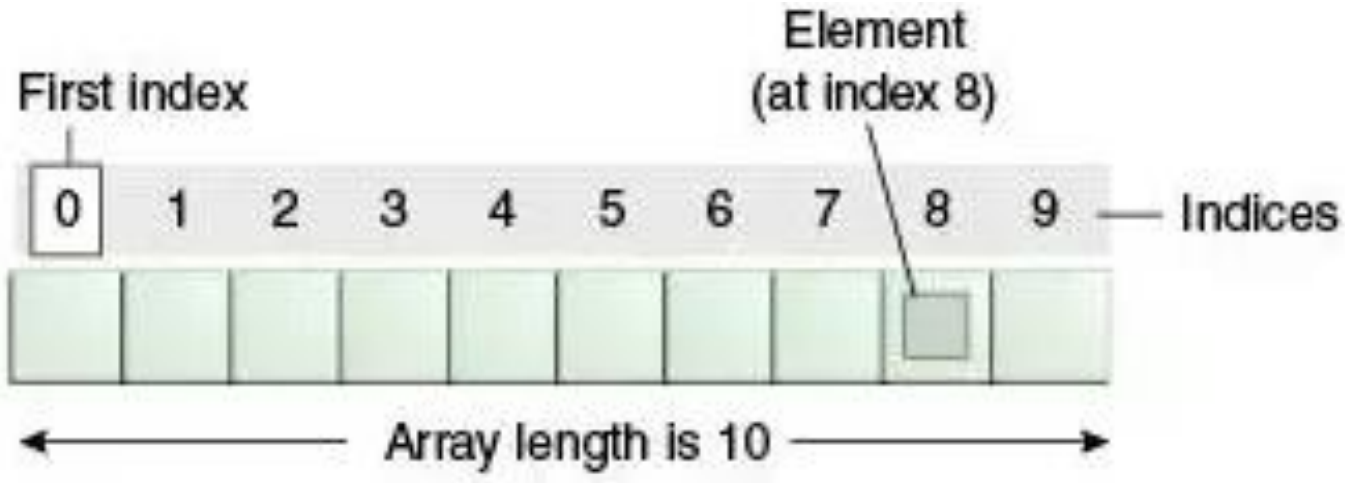
- It is a collection of values that have the same data type.
- Eg:
  - A collection of int data values or
  - A collection of bool data values
- All stored values are accessed using arrayname.
- To access a particular value stored in an array index is used.
  - The first array index is always 0
- All arrays consist of contiguous memory locations.

Types

- One dimensional array
- Two dimensional array (a table or a matrix)
- Multi-dimensional array

One Dimensional Array

- Declaration:
- Syn: type arrayName [arraySize];
  - arraySize must be an integer constant greater than zero.
  - type can be any valid C++ data type
- Eg: int A[10];

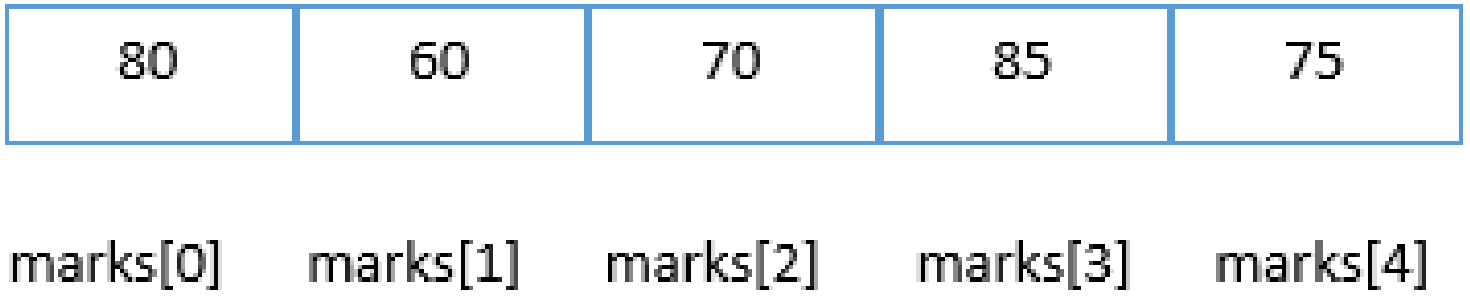


Initialization

- There are two types of initialization:
- 1. **Compile time initialization**
  - Can be Initialized during declaration
  - Or can be initialized during programming
- 2. **Run time initialization**
  - Initialized during the execution of the program

1. Compile Time Initialization

- int marks[5] ={80, 60, 70, 85, 75};



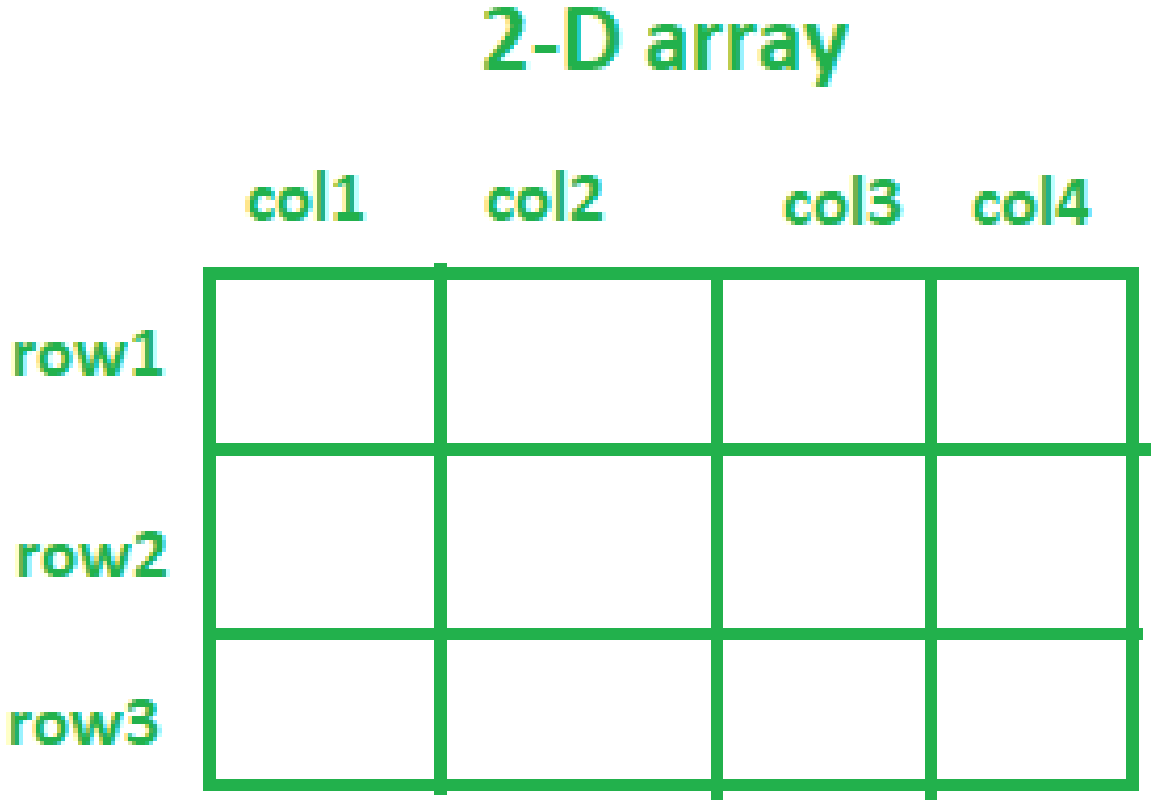
- int marks[]={56, 78, 69, 94, 88, 52};
- float price[6]={12.25, 36.75, 45.50};
- int A[3]={44, 33, 88, 99, 22};

2. Run Time Initialization

- It is done using 'for' loop.
- Eg:

```
int A[5];
for (i=0; i<=5; i++);
{
        cin>> A[i];
}
```

Two Dimensional Array



2D Array

	Col1	Col2	Col3	Col4	....
Row1	Arr[0][0]	Arr[0][1]	Arr[0][2]	Arr[0][3]	
Row2	Arr[1][0]	Arr[1][1]	Arr[1][2]	Arr[1][3]	
Row3	Arr[2][0]	Arr[2][1]	Arr[2][2]	Arr[2][3]	
Row4	Arr[3][0]	Arr[3][1]	Arr[3][2]	Arr[3][3]	
:					

- **Declaration:**
- Syn: type arrayname [rowsize][colsize];
- Eg: int A[2][3];
- Total no of elements = rowsize x colsize
- A[0][0]=1 A[1][0]=4
- A[0][1]=2 A[1][1]=5
- A[0][2]=3 A[1][2]=6

Initialization

- int A[2][3] = {1, 2, 3, 4, 5, 6};
- int A[2][3] = {{1, 2, 3}, {4, 5, 6}};
- int A[2][3]={1, 2, 3, 4};
- int A[2][3]={{1}, {2,3,4}};

2D Accessing

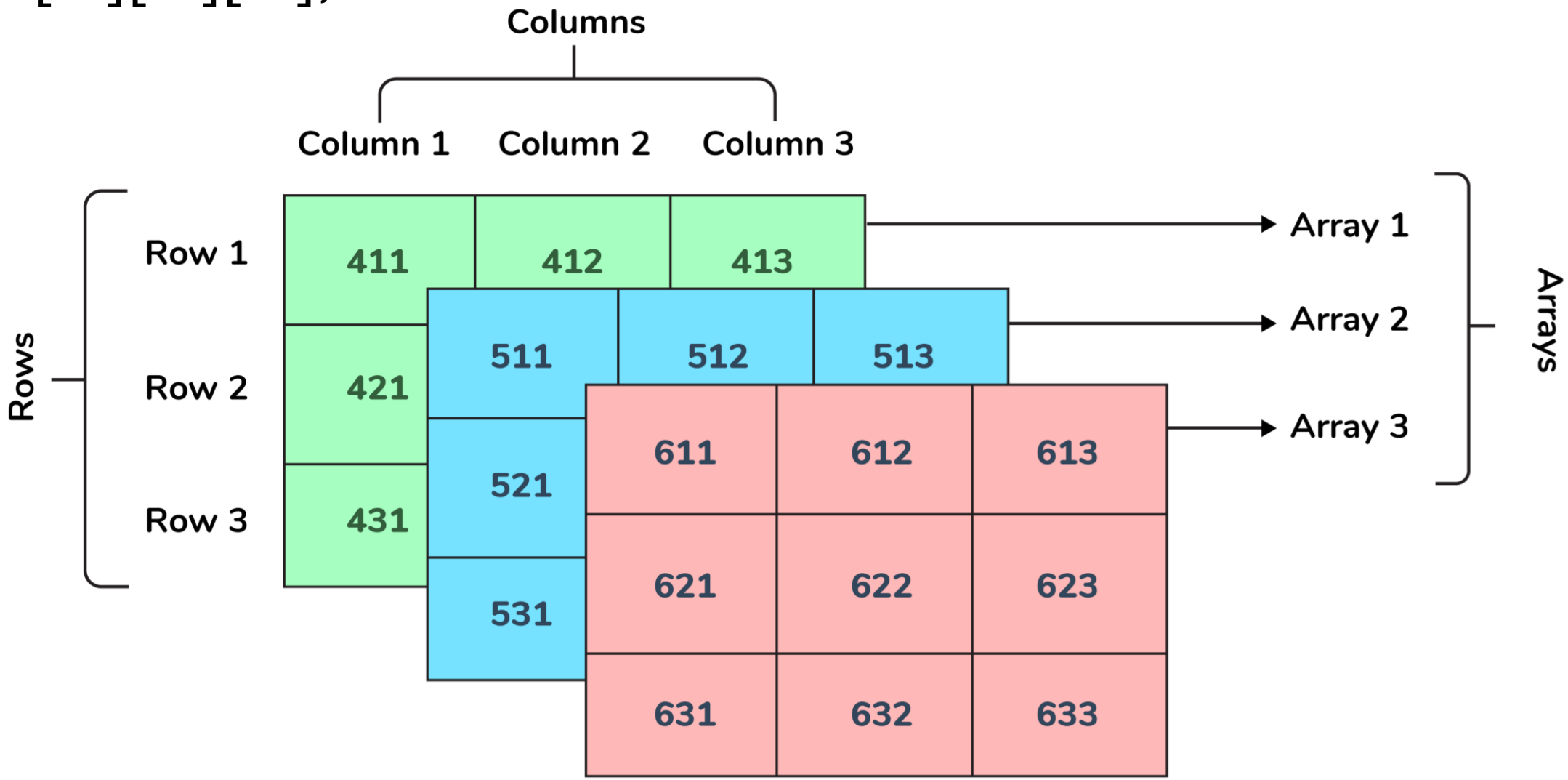
- It is done using 2 for loops.
- for(i=0; i<row; i++)
- {

```
    for(j=0; j<col; j++)
    {
        cin>>A[i][j];
    }
```

- }

Multi-Dimensional Array

- int A[s1][s2][s3];



Functions

- A function is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a function.
- C++ provides some pre-defined functions, such as main().
- Advantages:
  - Code reusability
  - Code optimization
- Types:
  - Library function
  - User-defined function

There are 3 parts:

- 1. Function Declaration / Function Prototype
- 2. Function call
- 3. Functions Definition

1. Declaring a Function

- To declare a function, specify the name of the function, followed by parentheses ():

Return_type Function_name ([arguments]);

- Where,
  - Return_type → Datatype of the value being returned to the calling function.
  - Function_name → Name of the function.
  - Arguments → Parameters being passed to the function.

2. Function Call

- Appears within the calling function.
- Syn: **Function_name ([arguments])**

Or

Var = Function_name ([arguments]);

3. Function Definition

- Inside the function (the body),we add code that defines what the function should do.
- It is written outside the main function.
- Syntax:

```
return_type Function_name ([arguments])
{
        // code to be executed
}
```

Example:

```
void Print();        // Function Declaration
int main()
{
        Print();     //Function Call
}

void  Print()        // Function Definition
{
        cout<<"Hello World.";
}
```

Types:

- 1. Function with no arguments and no return stmnt.
- 2. Function with no arguments and with return stmnt.
- 3. Function with arguments and no return stmnt.
- 4. Function with arguments and with return stmnt.
- 5. Recursive Function
- 6. Inline Function

Example1: No Arguments, No Return


```
#include <iostream>
using namespace std;

void func();
int main()
{
        func();
        func();
        func();
}
```

```
void func()
{
  static int i=0;       //static variable
  int j=0;    //local variable
  i++;
  j++;
  cout<<"i=" << i<<" and j=" <<j<<endl;
}
```

Example 2: No Arguments, With Return


```
#include<iostream>
using namespace std;

int SUM( )
{
        int a=15, b=28, sum;
        sum=a+b;
        return sum;
}

int main()
{
        int S = SUM();
        cout<<"Sum of two no's :"<< S;
        return 0;
}
```

Example 3: With Arguments, No Return


```
#include<iostream>
using  namespace std;

void SQR(int a)
{
 cout<<“Sqaure of”<< a<<“is: ”<<a*a;
}
```

```
int  main()
{
   int  n;
   cout<<“Enter a number:”;
   cin>>n;
    SQR(n);
   return 0;
}
```

Example 4: With Arguments, With Return


```
#include<iostream>
using namespace std;
long  SQR ( int );
int main()
{
    int n=12;
    long res;
    res = SQR(n);
    cout<<"Result is:"<<res;
    return 0;
}
```

```
long SQR (int a)
{
        return (a*a);
}
```

Functions

- The main() function:
  - Starting point of the execution.
  - Returns the value of type int by default, hence keyword int in main() is optional.

- INLINE function:
  - Expanded inline when it is invoked.
  - The compiler replaces the function call with the respective code.
  - Purpose – eliminate the cost of calls to small functions.

C++ Function Parameters

- Parameters act as variables inside the function.
- Parameters are specified after the function name, inside the parentheses.
- We can add as many parameters as we want, just separate them with a comma:
- Syntax:

```
void functionName( parameter1, parameter2, parameter3)
        {
         // code to be executed
        }
```

```
#include <iostream>
#include <string>
using namespace std;
void myFunction ( string fname)
{       cout << fname << " Surname\n";  }

int main()
{       myFunction("Liam");
        myFunction("Jenny");
        myFunction("Anja");
        return 0;   }
```

Liam Surname

Jenny Surname

Anja Surname

- When a parameter is passed to the function, it is called an argument.
- So, from the example above:
  - fname is a parameter,
  - while Liam, Jenny and Anja are arguments.

C++ Default Parameters

- Assign default parameter value, by using the equals sign (=).

```
#include <iostream>
#include <string>
using namespace std;
void myFunction(string country = "Norway")
 {
    cout << country << "\n";
 }
int main()
{
myFunction("Sweden");
myFunction("India");
myFunction();
myFunction("USA");
return 0;
}
```

- A parameter with a default value, is often known as an "optional parameter".
- From the example above, country is an optional parameter and "Norway" is the default value.

```
void myFunction(string country = "Norway")
```

```
#include <iostream>
#include <string>
using namespace std;
void myFunction(string fname, int age)
{
 cout << fname << " Surname. " << age << " years old. \n";      }

int main()
{       myFunction("Liam", 3);
        myFunction("Jenny", 14);
        myFunction("Anja", 30);
        return 0;     }
```

Liam Surname. 3 years old.

Jenny Surname. 14 years old.

Anja Surname. 30 years old.

Note: Inside the function, we can add as many parameters as you want

Pass by Reference

- we used normal variables when we passed parameters to a function.
- We can also pass a reference to the function. This can be useful when we need to change the value of the arguments:

```
#include <iostream>
using namespace std;

void swapNums(int &x, int &y)
 {
        int z = x;
        x = y;
       y = z;
}
```

```
int main()
{
    int n1 = 10;
    int n2= 20;
    cout << "Before swap: " << "\n";
    cout << n1 << n2 << "\n";
     swapNums (n1,n2);
     cout << "After swap: " << "\n";
     cout << n1 << n2 << "\n";
     return 0;
}
```

Before swap:

1020

After swap:

2010

Difference

Call by Value

1. A copy of value is passed to the function
2. Changes made inside the function is not reflected on other functions
3. Actual and formal arguments will be created in different memory location

Call by Reference

1. An address of value is passed to the function
2. Changes made inside the function is reflected outside the function also
3. Actual and formal arguments will be created in same memory location

Pass Arrays as Function Parameters


```
#include <iostream>
using namespace std;
void printArray(int arr[5]);
int main()
{
    int arr1[5] = { 10, 20, 30, 40, 50 };
    int arr2[5] = { 5, 15, 25, 35, 45 };
    printArray(arr1);
    printArray(arr2);
}
```

```
void printArray(int arr[5])
{
    cout << "Printing array elements:\n“;
    for (int i = 0; i < 5; i++)
    {
        cout<<arr[i]<<"\n";
    }
}
```

- The function myFunction() takes an array as its parameter (int arr[5]), and loops through the array elements with the for loop.
- When the function is called inside main(), we pass along the arr array, which outputs the array elements.
- Note that when you call the function, you only need to use the name of the array when passing it as an argument myFunction(arr).
- However, the full declaration of the array is needed in the function parameter (int arr[5]).

C++ Function Overloading

- Multiple functions with the same name but with different parameters.

```
#include <iostream>
using namespace std;
int Sum (int x, int y)
 {
  return x + y;
}
double Sum (double x, double y)
 {
  return x + y;
}
int main()
{
     int Num1 = Sum (8, 5);
     double Num2 = Sum(4.3, 6.26);
     cout << "Int: " << Num1<< "\n";
      cout << "Double: " << Num2;
      return 0;
}
```

Int: 13

Double: 10.56

- Note: Multiple functions can have the same name as long as the number and/or type of parameters are different.

End of Unit I

BASIC CONCEPTS IN OOP

- Classes - Basic template for creating objects.
- Objects - Basic run time entities.
- Data Abstraction & Encapsulation - Wrapping data and functions into single unit.
- Inheritance - Properties of one class can be inherited into others.
- Polymorphism - ability to take more than one forms.
- Dynamic Binding - code which will execute is not known until the program runs.
- Message Passing - Object.message (Information) call format.