ARRAYS AND STRINGS IN C

UEM Kolkata

CA dept

TOPICS

- Arrays - Introduction
- Need of Arrays
- Characteristics of Arrays
- Types of Arrays
- Why indexing of arrays start at 0 (zero)

NEED OF ARRAYS

- An ordinary variable is capable of storing only one value at a time

- However, there are situations in which we would be requiring to store more than one value at a time in a single variable

NEED OF ARRAYS

- e.g. Store the % marks obtained by 100 students
- Two options to store the marks are:
  - Construct 100 variables to store % marks obtained by 100 different students i.e. Each variable containing one student‟s marks
  - Construct one variable (called a subscripted variable) capable of storing all the hundred marks

NEED OF ARRAYS

```
#include<stdio.h>
void main()
{
    int marks1, marks2, marks3, marks4, marks5;
    scanf("%d",&marks1);
    scanf("%d %d %d"%d",&marks2, marks3,marks4,
    marks5);
}
```

NEED OF ARRAYS

marks1		marks2		marks3	
marks4		marks5		1010	1011
1012	1013	1014	1015	1016	1017

ARRAY

- General Definition
  - "Array is an ordered collection of homogeneous elements"

- **Collection is ordered**
- **Elements are homogeneous**

ARRAY

- **Collection is ordered**
  - items added to the collection are maintained in the order it were added to the collection

- **Elements are homogeneous**
  - elements are of same type, group or class

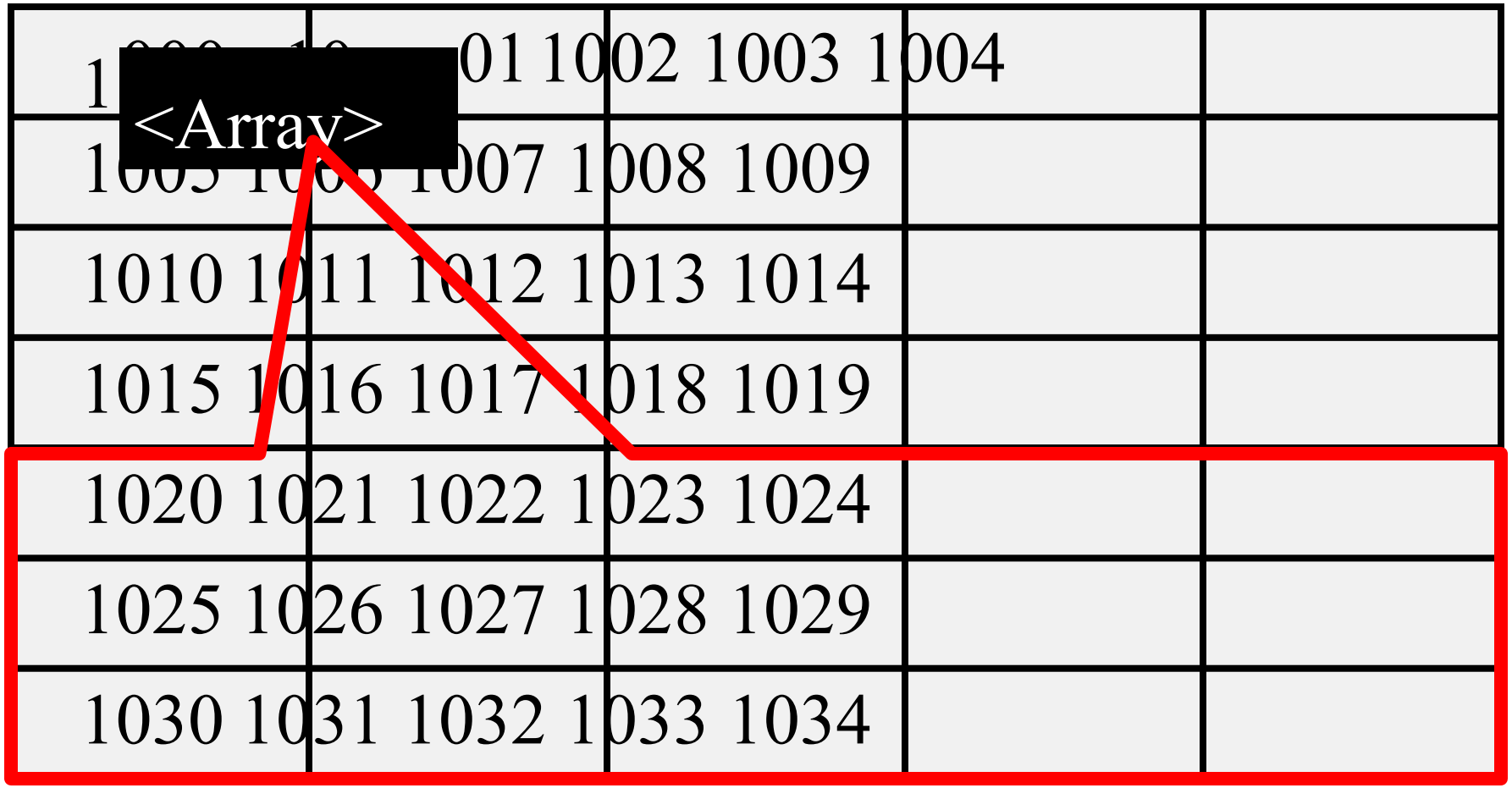
ARRAYS IN C

- Array is a collection of same type elements under the same variable identifier referenced by index number
- Arrays
  - Are a sequence of memory locations referred to by a common name
  - Can store multiple values of the same data type
- The individual data items are called **elements** of the array

VARIABLES IN MEMORY

1000	1001	1 char c= „A“; 3		1004
=10; 006		1007	1008	1009
		1012	1013	1014
1015	1016	1017	1018	1019
1020	1021	1022	1023	1024
1025	1026	1027	1028	1029
1030	1031	1032	1033	1034

ARRAYS IN MEMORY



ARRAYS IN C

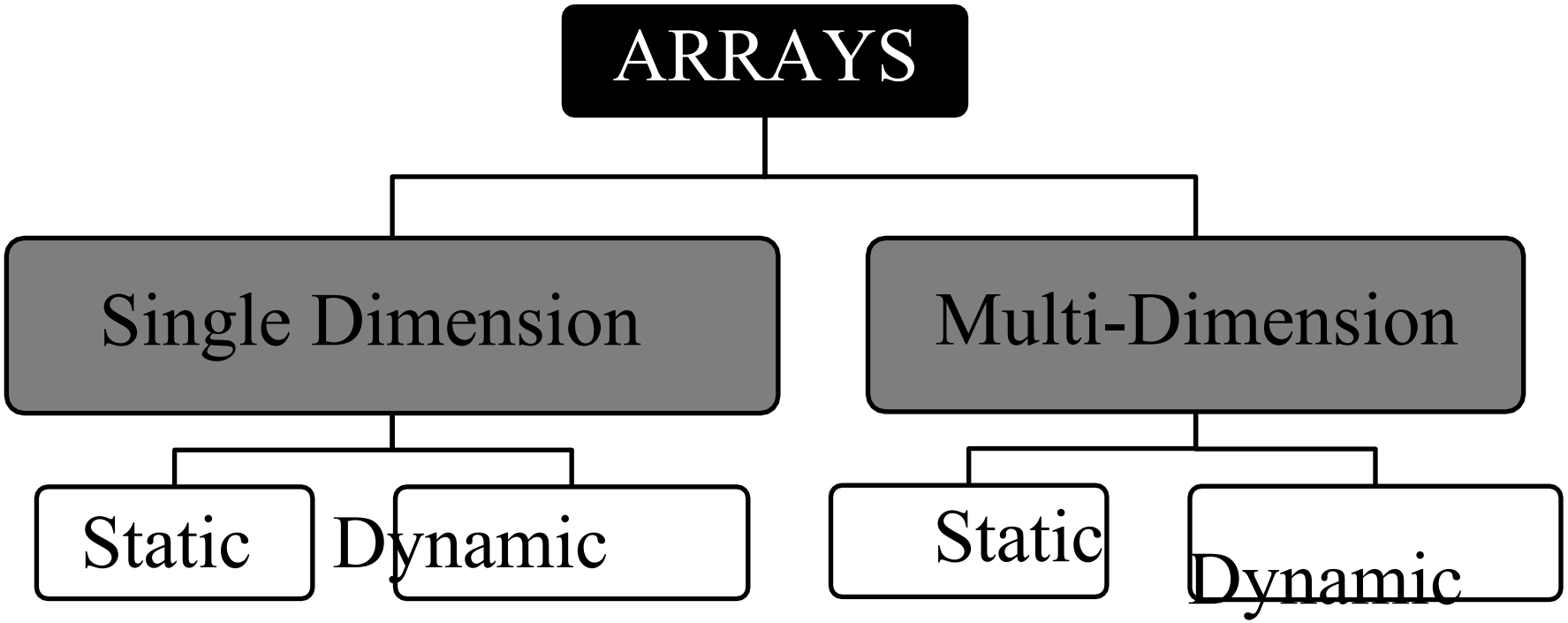
- **Array data structure**, an arrangement of items at equally spaced addresses in computer memory

- **Array data type**, used in a programming language to specify a variable that can be indexed

CHARACTERISTICS OF ARRAYS IN C

- Array is a **subscripted variable**
- **Random access** via indexing operator []
- Indexing
  - **starts at 0**
  - Ends at N-1 (where N is maximum the array size)
- The name of the **array refers to the address of the first element** in the array

TYPES OF ARRAYS



STATIC vs DYNAMIC

- Static arrays have their sizes declared from the start and the size cannot be changed after declaration

- Dynamic arrays that allow you to dynamically change their size at runtime, but they require more advanced techniques such as pointers and memory allocation.

DECLARING SINGLE DIMENSION ARRAYS

- SYNTAX:

<data type> <variable name>[<dimension size>]

  - **data type** specifies the data type of the values that will be stored in the array
  - **dimension size** indicates the maximum number of elements that can be stored in the array

- The name of the array refers to the address of the first element in the array

DECLARING SINGLE DIMENSION ARRAYS

- Example:

  <data type> <variable name>[<dimension size>]

```
int num[5];
```

- This declares an array **num** of size 5
- Elements of the array are **num[0]**, **num[1]**, **num[2]**, **num[3]** and **num[4]**
- 0, 1, 2, 3 and 4 are called subscripts or indexes
- The array name **num** refers to the address of **num[0]** i.e. first element of the array

INITIALISING ARRAYS

- Initialisation of an array
  - Refers to assigning values to the elements of the array
  - Can happen along with declaration or after declaration

SINGLE DIMENSION ARRAYS

17		2		3	
44		15		1005 1010	1011
1012	1013	1014	1015	1016	1017

```
int num[5]={1    ,2,3,44,   15};
        7

printf("%d",num[2]);
```

INITIALIZING ARRAYS

```
#include<stdio.h>
void main()
{
        int marks[5]={34,21,33,12,15};
}
```

```
#include<stdio.h>
void main()
{
        int marks[]={34,21,33,12,15};
}
```

INITIALIZING ARRAYS

```
#include<stdio.h>
void main(){
        int marks[5];
        marks[0]=34;
        marks[1]=21;
        marks[2]=33;
        marks[3]=12;
        marks[4]=15;
}
```

```
marks[0]=34;
marks[3]=12;
marks[4]=15;
marks[1]=21;
marks[2]=33;
```

READING ARRAYS

```
#include<stdio.h>
void main()
{
        int marks[5], int i=0;
        for(i=0;i<5;i++)
        {
                scanf("%d",&marks[i]);
        }
}
```

ARRAYS – MEMORY MAP

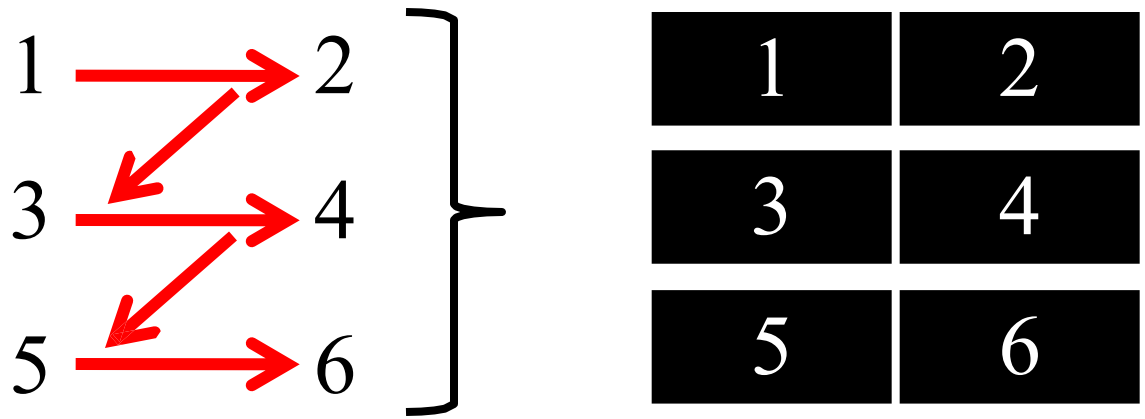
34		21		33	
12		15		1010	1011
1012	1013	1014	1015	1016	1017

MULTI-DIMENSIONAL ARRAYS

- Don‟t really exist in C
- C allows “arrays of arrays”
- Use one set of bracket for each dimension
- Can be initialized with nested initializer lists

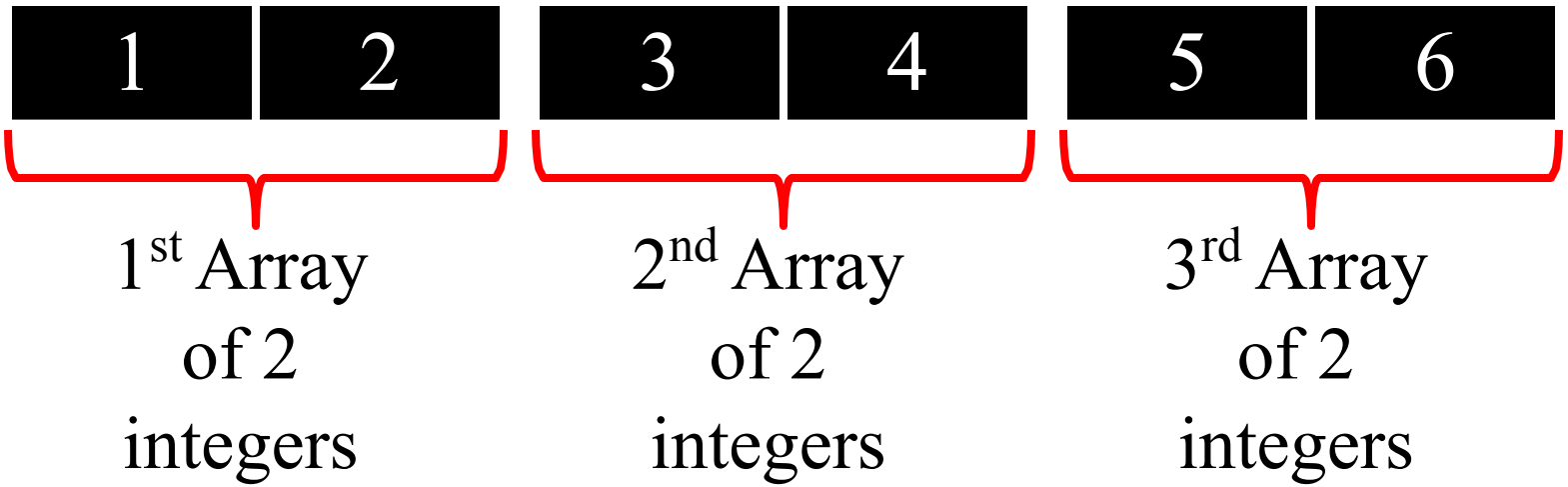
MULTI-DIMENSIONAL ARRAYS

- Consider the following 3 x 2 array:



- C stores it linearly: 1 2 3 4 5 6

MULTI-DIMENSIONAL ARRAYS



- The compiler interpret it as:

  an array of **"3 arrays of 2 integers"**

C allow "arrays of arrays"

DECLARING ARRAYS

- Example:

<data type> <variable name>

[<dimension 1 size>] [<dimension 2 size>]....

[<dimension N size>]

```
int num[3][2];
```

- This declares an array **num** of size 3 x 2
- Elements of the array are **num[0][0]**, **num[0][1]**, **num[1][0]**, **num[1][1]**, **num[2][0]** and **num[2][1]**

INITIALISING ARRAYS

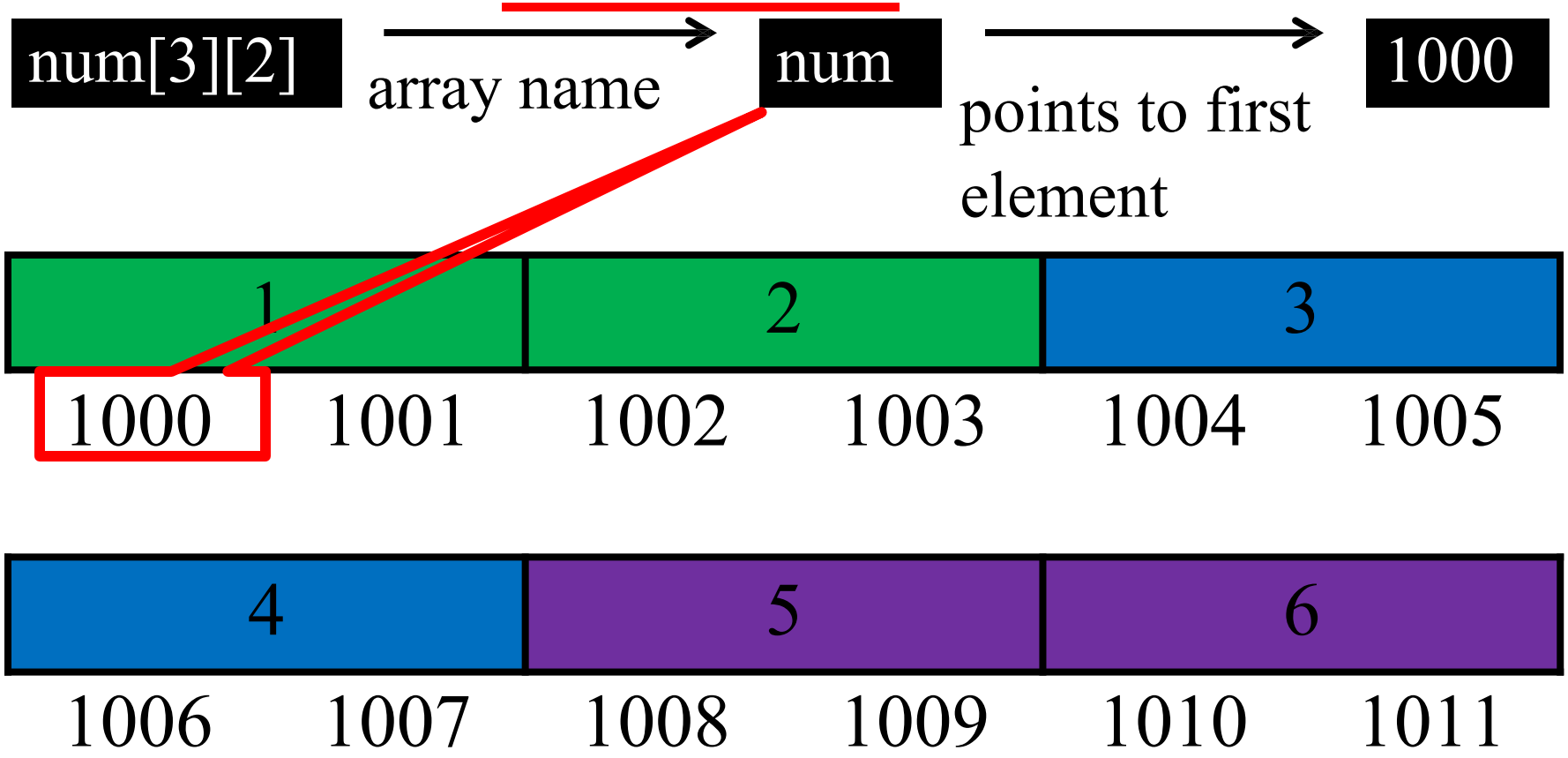
1	2 3	
4	5 6	

10/10/2013

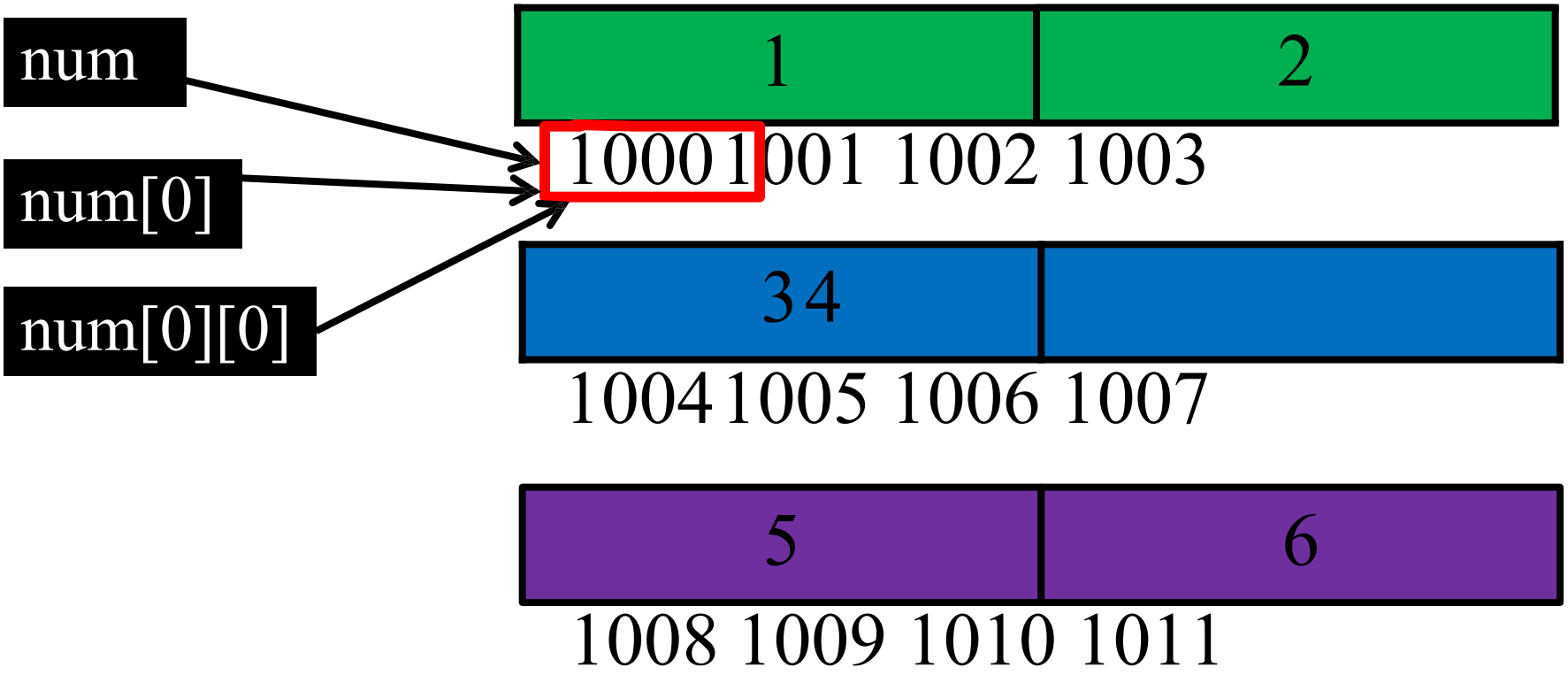
```
int num[3][2]={1,2,3,4,5,6};
```

```
int num[3][2]={
                {1,2},
                {3,4},
                {5,6}
                };
```

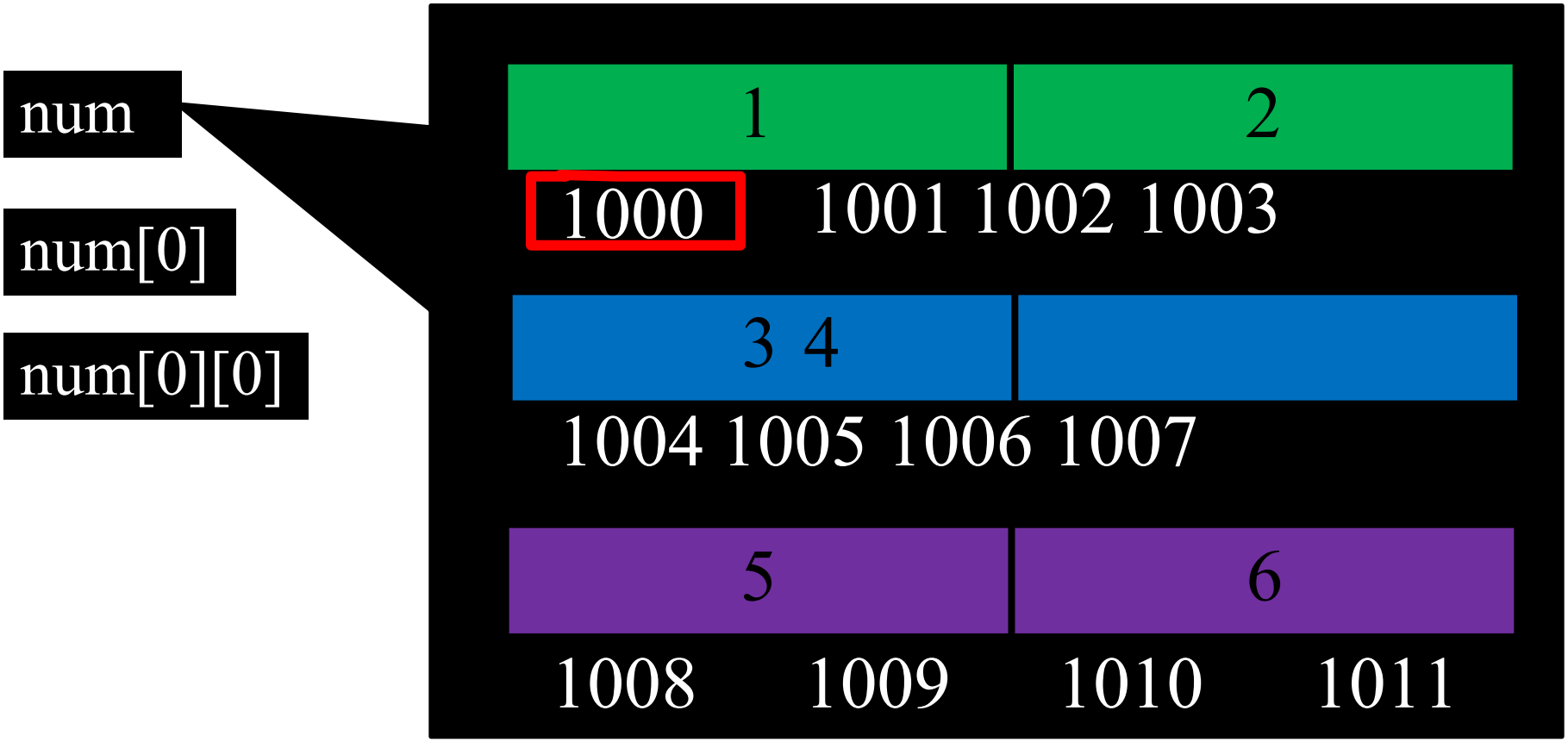
INITIALISING ARRAYS



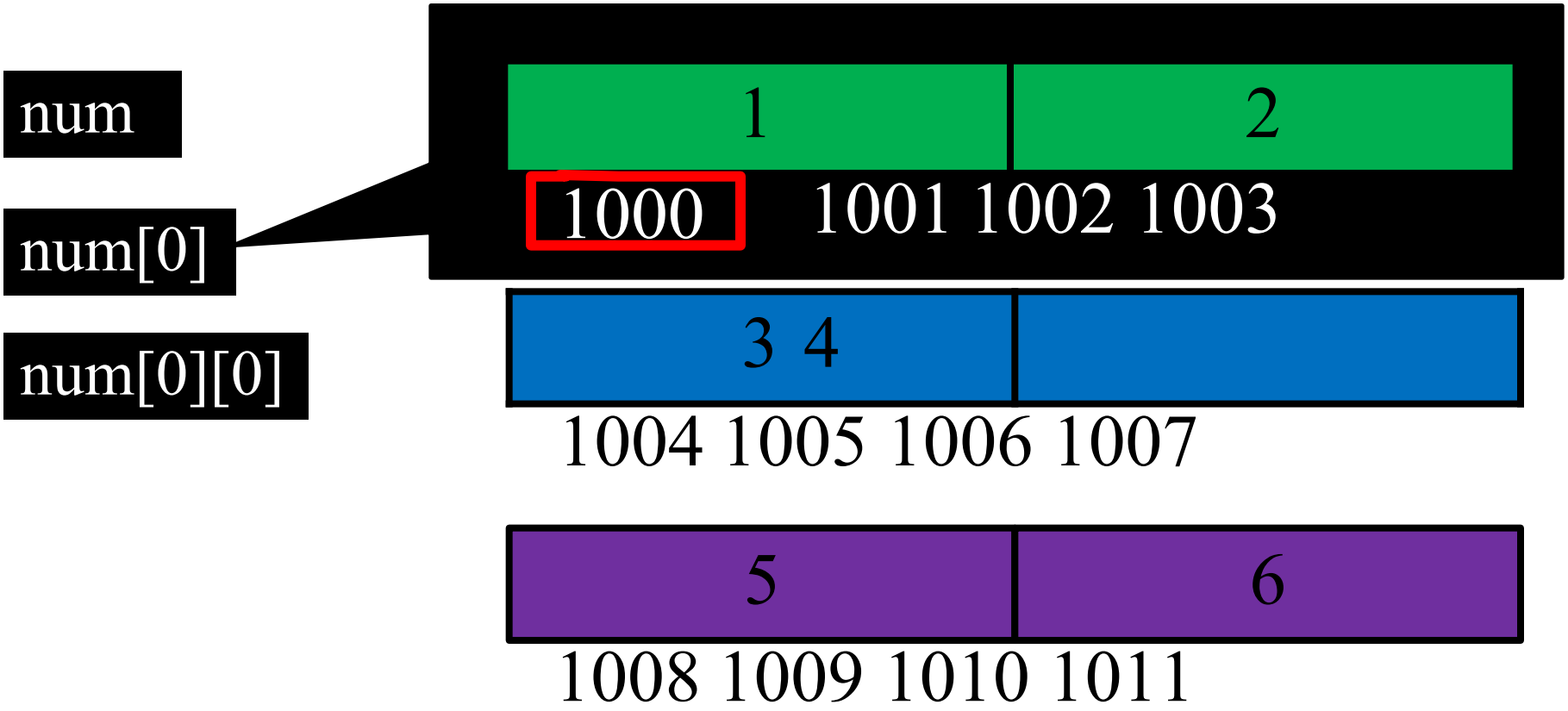
INITIALISING ARRAYS



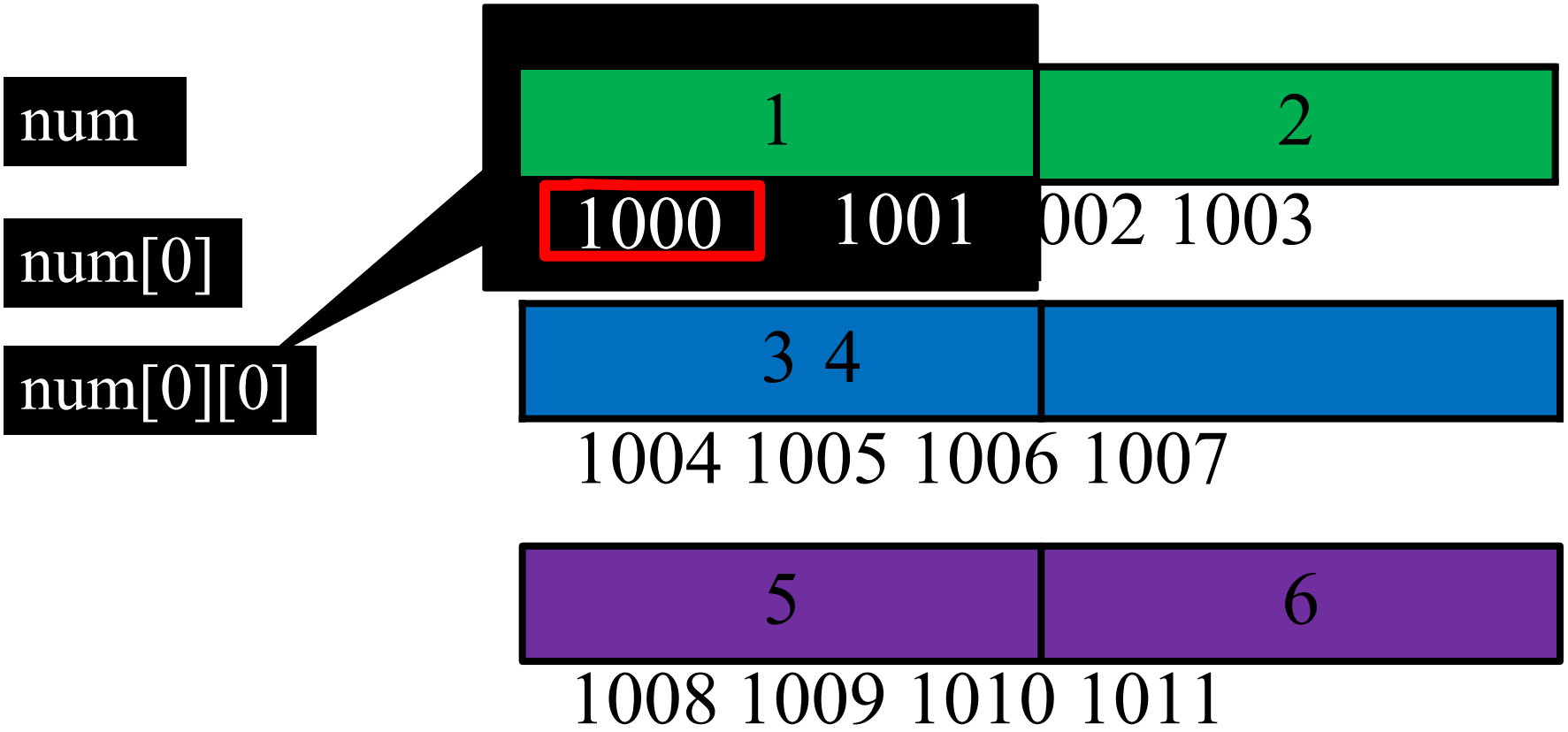
INITIALISING ARRAYS



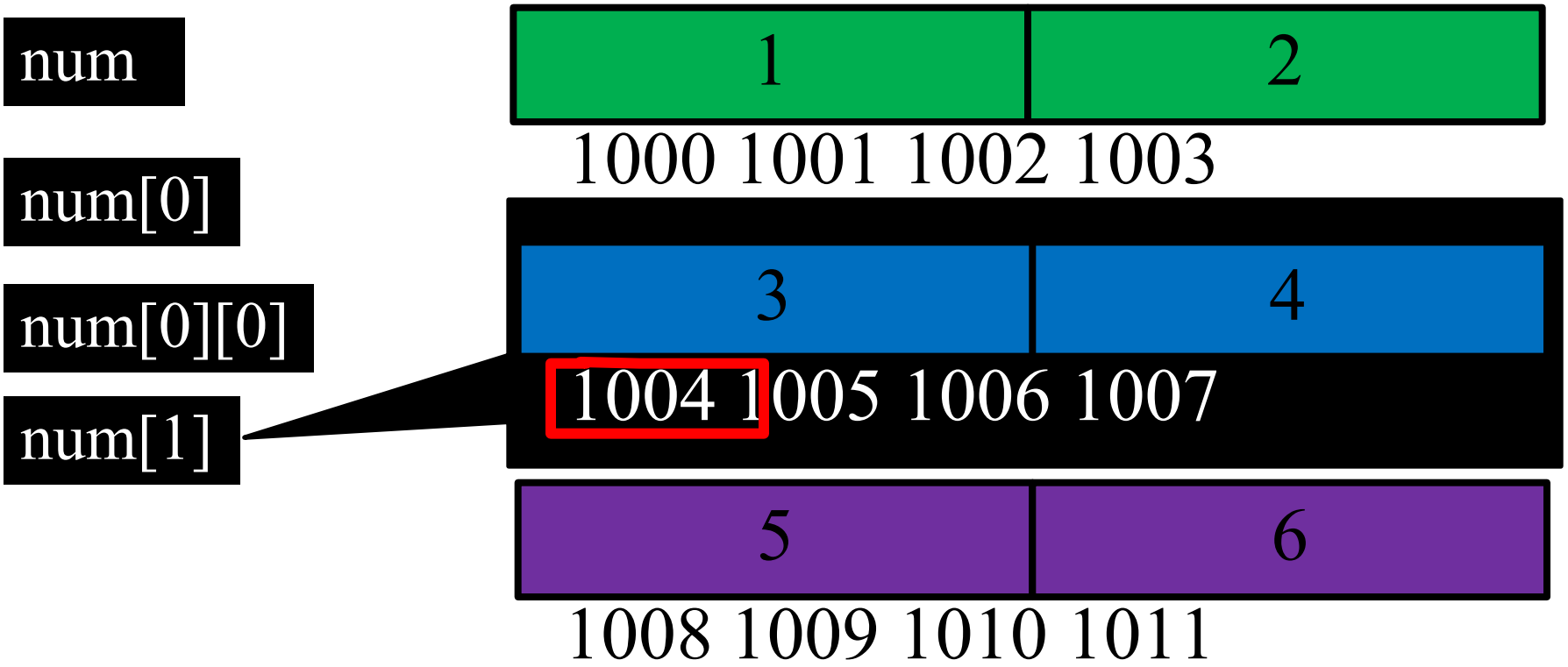
INITIALISING ARRAYS



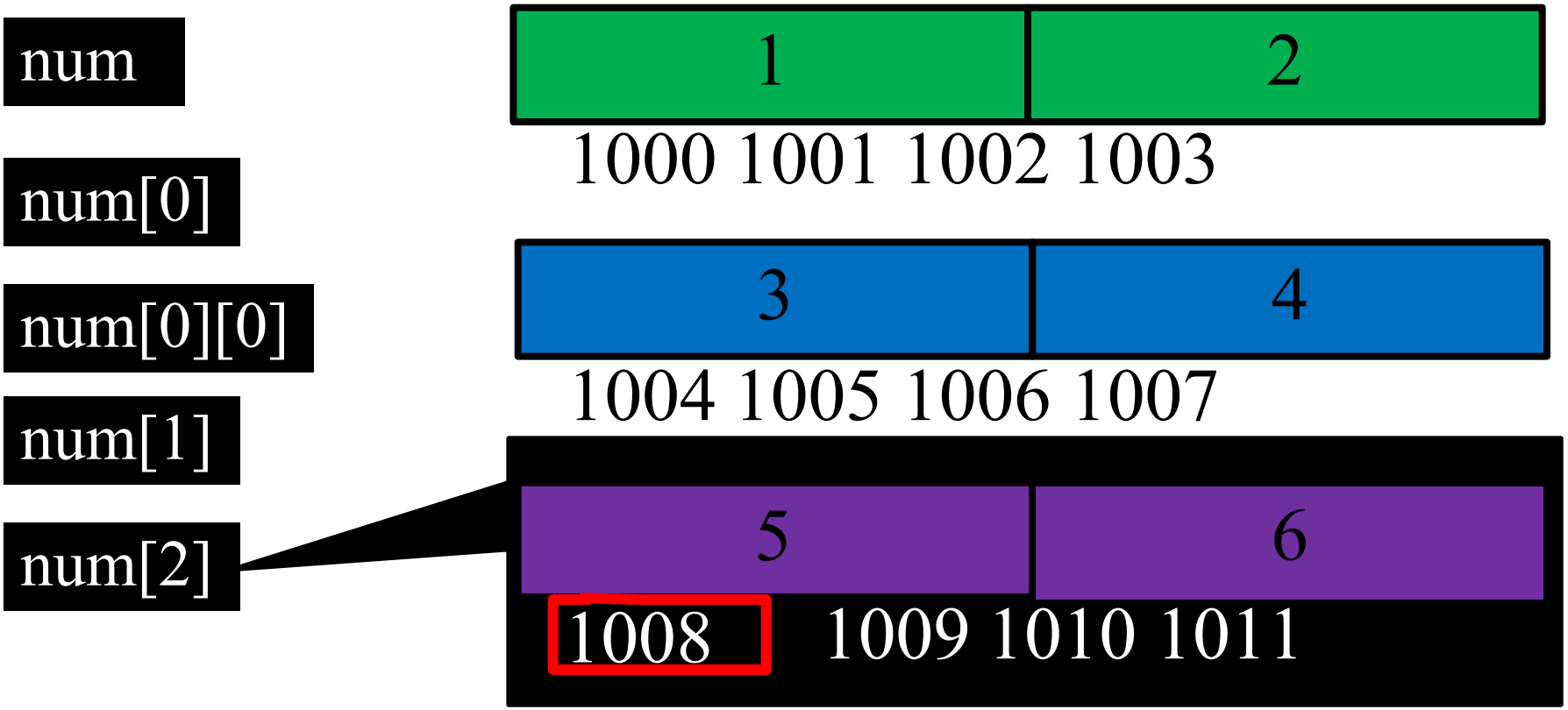
INITIALISING ARRAYS



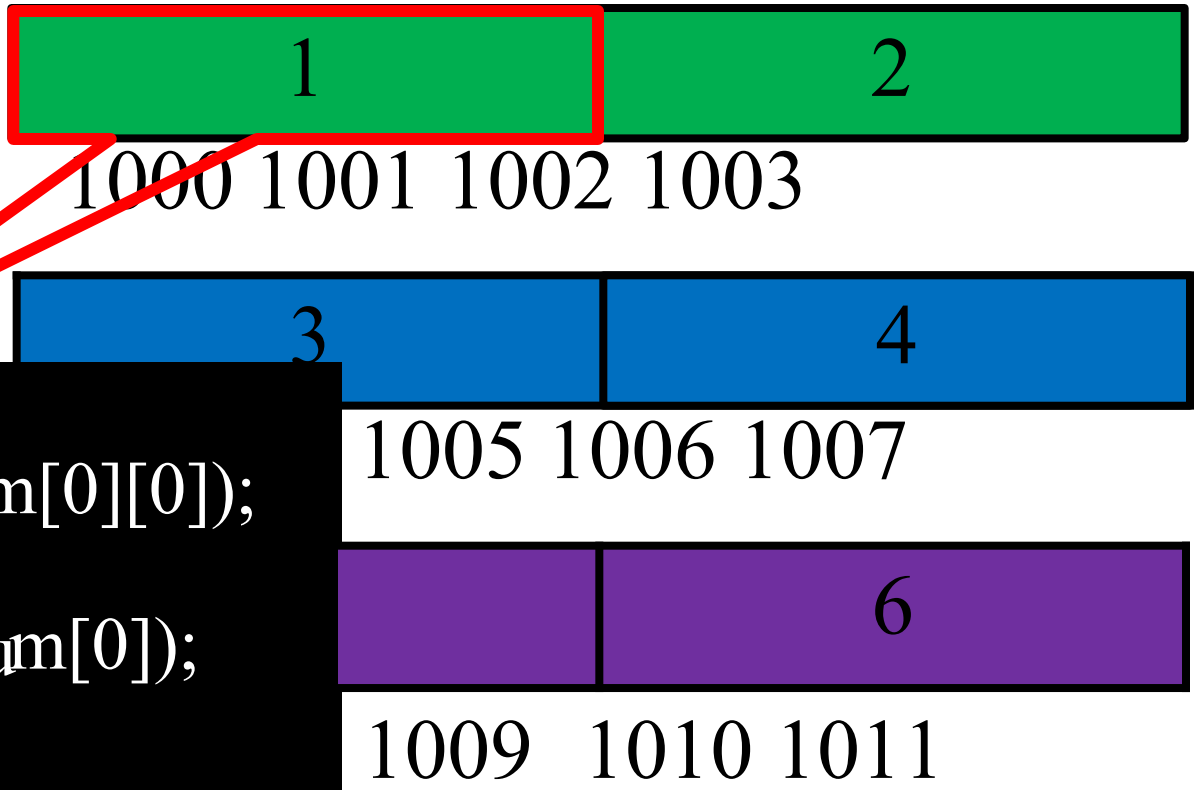
INITIALISING ARRAYS



INITIALISING ARRAYS



ACCESSING ARRAYS




```
printf("%d",num[0][0]);

printf("%d",*num[0]);

printf("%d",**num);
```

TWO DIMENSIONAL ARRAY

```
#include<stdio.h>
void main(){
        int marks[3][2], int i=0,j=0;
        for(i=0;i<3;i++){
                for(j=0;j<2;j++){
                        scanf("%d",&marks[i][j]);
                }
        }
}
```

DECLARING STRING

- Example:

<data type> <variable name>[<dimension size>]

```
char str[10] ;
```

- Creates a character array **str** that can store a maximum of 9 characters
- The last element of a character array must always be the null character or string terminator [**\0**]
- It indicates the end of the character array
- The null character cannot be printed

INITIALISING ARRAYS

```
char str[6];
str[0] = „I“;
str[1] = „N“;
str[2] = „D“;
str[3] = „I“;
str[4] = „A“;
Str[5] = „\0“;
```

```
char str[ ] = “INDIA”;
```

Both statements assign the name **INDIA** to the array **str**

I
N
D
I
A
„\0“

PRINTING ARRAYS

- All elements of a character arrays can be printed using a single **printf()** function
- The format code for printing a string is %s

```
printf("%s", str);
```

✔ The above statement prints the value of the character array **str**

✔ **The printf() function displays all the characters in the array until the null character „\0" is reached**

INITIALISING ARRAYS

I	N	D	I	A
„\0“	1006	1007	1008	1009
1010	1011	1012	1013	1014

EFFECT OF NULL CHARACTER „\0"ON STRING

```
void main(){
    char
    name[5]={„i",“n",“d",“i",“a"};
    char name1[5]=“india”;
    printf(“NAME = %s”,name);
    p[illegible]%s”,name)
}
```

First Execution

```
NAME  =  INDIA
NAME1 =  INDIA
```

Second Execution

```
NAME  =  INDIAä
NAME1 =  INDIA♥INDIAä
```

STRING FORMAT WITH PRECISION

```
main()
{
   char name[6]={'I','N','D','I','A'};
   printf("\n%.2s",name);
   printf("\n%.10s",name);
   printf("\n%10.4s",name);
   printf("\n%10s",name);
   printf("\n%-10s",name);
}
```

```
IN
INDIA
      INDI
     INDIA
INDIA
```

CHARACTER ARRAY vs STRING

```
main(){
    char text[]="have a nice day";
    int i=0;
    while(i<=15)
    {
        printf("%c",text[i]);
        i++;
    }
    printf("%s",text);
}
```

```
have a nice day
have a nice day
```

WHY ARRAY STARTS AT 0

char str[ ] = “INDIA”;p

str[3]);

Index	str	Address
0	I	100
1	N	101
2	D	102
3	I	103
4	A	104
5	„\0“	105

The name of the array refers to the address of the first element in the array

str[3]

*(str+3)

*(100+3)

*(103)

WHAT IF ARRAY STARTS AT 1

```
char str[ ] = "INDIA";
```

```
printf("%c",str[4]);
```

Index	strAddress	
1	100	I
2	101	N
3	102	D
4	103	I
5	104	A
6	105	„\0"

str[4]

↓

*(str+4)

↓

*(100+4)

↓

*(104)

SUGGESTED READING

- BOOKS:
  - Yashwant Kanetkar, **"Let Us C"**, 5th Edition, BPB Publications
  - Chuck Allison, "**Thinking in C**", Mindview Inc
- ONLINE RESOURCES
  - Microsoft MSDN library

Thank You