Chapter 06
Data type
User-defined Data Type

Dr. Le Thanh Sach

Content

- Typedef
- Struct
- Array

Convention

USR_DT = User-defined data type

Typedef

- typedef is a keyword used in C++ language to assign alternative names to existing types. It’s mostly used with user defined data types, when names of data types get slightly complicated.
  - The new name is more understandable, in the context of the problem.
  - Write code shorter
  - Can be used like fundamental data type

Typedef

```
#include <iostream>
using namespace std;
/*new name for "unsigned byte"*/
typedef unsigned char byte;
int main(){
byte a = 78;
unsigned char b = 'A', c;
c = a; a = b;
cout << "a = " << a << endl;
cout << "b = " << b << endl;
cout << "c = " << c << endl;

system("pause");
return 0;
}
```

Typedef

- Example
  - New defined data type “byte” can be used as type instead of using “unsigned byte”
    - => Increase the meaning of “unsigned char”
    - => Your code is nicer and shorter
    - => Can be used with origin data type
      - Variable a (of new data type) can be assigned to variable c (origin data type)
      - Variable a (of new data type) can receive value from variable b (origin data type)
      - Can print variable a (of new data type) like a number or a character
      - Variable a can be used in an expression where operands use origin data type

Typedef

- Or we can use typedef to:
  - Define new name for enum type
  - Define new name for struct

Struct

- Why do we need struct?
  - Problem: student management system
    - Program needs to store information for each student:
      - Identifier
      - Name
      - Date of birth
      - Address
      - Phone number
      - Email
      - ….

Struct

- Why do we need struct?
  - Problem: Student management system
    - If we use built-in data types to store information of students in memory
      - Need MANY variables where each variable represents information of a student.
      - => Inconvenient: ugly code, hard to understand, etc.
      - => Even if we only store information of some students in memory: variable declaration lines occupy a large area of source code.

Struct

- Why do we need struct ?
  - Problem: Student management system
  - Another similar problems
    - Information of a point or a vector
    - Information of a product, goods in supermarket
    - ...
  - Solution
    - GATHER all related data into one block
      - Are always allocated contiguously in memory
      - Are always released from memory together
      - Allow different component data fragments can be retrieved independently by its name

Struct

- Why do we need struct?
  - Solution
    - GATHER all related data into one block
      - Are always allocated contiguously in memory
      - Are always released from memory together
      - Allow different component data fragments can be retrieved independently by its name
    - In C: struct is used
    - In C++: class is used

Struct

- What is struct?
  - Is a composite data type consisting of partial, built-in data types. Partial types can be the same type or they can be different. They can also be struct.
  - In object oriented programming languages, a similar data type but with more features (Class) can be used instead of struct.

Struct

- What is struct?
  - Example

```
struct sStudent{
char id[5];
char name[50];
float gpa;
};
struct sPoint3D{
float x, y, z;
};
struct sVector3D{
float x, y, z;
};
struct sTable{
char code[10];
float width, length, height;
};
```

Struct

- Struct “sStudent”
  - Gather the relevant components (field) to describe a student
  - Data of each student contains:
    - id, name: Identifier and name of the student
      - Data type: array (will be covered in another chapter)
    - gpa: grade point average:
      - Data type: float
  - Always, CONTIGUOUS (ADJACENT) memory locations are used to store structure members in memory (struct “sStudent” in this case).

```
struct sStudent{
char id[5];
char name[50];
float gpa;
};
```

Struct

- Struct “sPoint3D” and “sVector3D”
  - Gather the relevant components (field) to describe a point and a vector in three dimensions.
  - Name of each component:
    - x,y,z: coordinates of point or vector
      - Data type: float or double
  - Each time the system allocates memory for a point or vector, it allocates a contiguous blocks for all the data points and vector

```
struct sPoint3D{
float x, y, z;
};
struct sVector3D{
float x, y, z;
};
```

Struct

- How to declare and use struct?

```
#include <iostream>
using namespace std;
struct sStudent{
    char id[5];
    char name[50];
    float gpa;
};
int main(){
    struct sStudent   s1;
    struct sStudent   s2 = {"001", "Nguyen Van An"};
    struct sStudent   s3 = {"001", "Nguyen Van An", 9.5f};

    cout << "ID:\t" << s3.id << endl;
    cout << "NAME:\t" << s3.name << endl;
    cout << "GPA:\t" << s3.gpa << endl;
    return 0;
}
```

Struct

- How to declare and use struct ?

```
#include <iostream>
using namespace std;
struct sStudent{
char id[5];
char name[50];
float gpa;
};
int main(){
struct sStudent    s1;
struct sStudent    s2 = {"001", "Nguyen Van An"};
struct sStudent    s3 = {"001", "Nguyen Van An", 9.5f};

cout << "ID:\t" << s3.id << endl;
cout << "NAME:\t" << s3.name << endl;
cout << "GPA:\t" << s3.gpa << endl;
return 0;
}
```

Define struct sStudent

Declare variables s1, s2, s3 with struct sStudent

S1: not assigned value

s2: incomplete initalization

s3: complete initialization

Struct

- How to declare and use Struct ?

```
#include <iostream>
using namespace std
struct sStudent{
char id[5];
char name[50];
float gpa;
};
int main(){
struct sStudent    s1;
struct sStudent    s2 = {"001", "Nguyen Van An"};
struct sStudent    s3 = {"001", "Nguyen Van An", 9.5f};

cout << "ID:\t" << s3.id << endl;
cout << "NAME:\t" << s3.name << endl;
cout << "GPA:\t" << s3.gpa << endl;
return 0;
}
```

Retrieve component data by name
Usage: <variable name>.<component name>

Struct

- Another example:

```
#include <iostream>
using namespace std;
struct sPoint3D{
float x, y, z;
};
int main(){
struct sPoint3D p1;
struct sPoint3D p2 = {1.5f, 2.5f, 3.5f};
p1.x = 1.0f; p1.y = 2.0f; p1.z = 3.0f;

cout << "p1 = (" << p1.x << "," << p1.y << "," << p1.z << ")" <<
endl;
cout << "p2 = (" << p2.x << "," << p2.y << "," << p2.z << ")" <<
endl;
return 0;
}
```

Note: declare and assign values

Notice how the printing was done

Struct

- Use typedef with struct
  - Remove keyword “struct” when declaring variables with type struct

```
#include <iostream>
using namespace std;
typedef struct sPoint3D{
float x, y, z;
} Point3D;

int main(){
struct sPoint3D p1 = {1.0f, 2.0f, 3.0f};
Point3D p2 = {1.0f, 2.0f, 3.0f};
cout << "p1 = (" << p1.x << "," << p1.y << "," << p1.z << ")" <<
endl;
cout << "p2 = (" << p2.x << "," << p2.y << "," << p2.z << ")" <<
endl;
}
```

Note: use typedef to define new data type - sPoint3D

Struct

- Use typedef with struct
  - Remove keyword “struct” when declaring variables with type struct

```
#include <iostream>
using namespace std;
typedef struct sPoint3D{
float x, y, z;
} Point3D;

int main(){
struct sPoint3D p1 = {1.0f, 2.0f, 3.0f};
Point3D p2 = {1.0f, 2.0f, 3.0f};
cout << "p1 = (" << p1.x << "," << p1.y << "," << p1.z << ")" <<
endl;
cout << "p2 = (" << p2.x << "," << p2.y << "," << p2.z << ")" <<
endl;
}
```

Note: use typedef to define new data type - sPoint3D

Notes: remove keyword “struct”

Array

Content

- Why do we need array?
- What is array?
- 1D Array
  - 1D array declaration
  - Read and write element
    - One element
    - All element
  - Applications
- 2D Array
  - 2D array declaration
  - Read and write element
    - One element
    - All elements
  - Applications
- String

Why do we need array ?

- Problem: Student management system
  - Suppose you want to store N students in memory and only use fundamental data types

  - Need N x M variables
    - M is number of attributes for each student
    - N = 100 students, M = 10 attributes
      - **=> 1000 variables!**

  - Possible but unreasonable!
    - Hard to read and develop

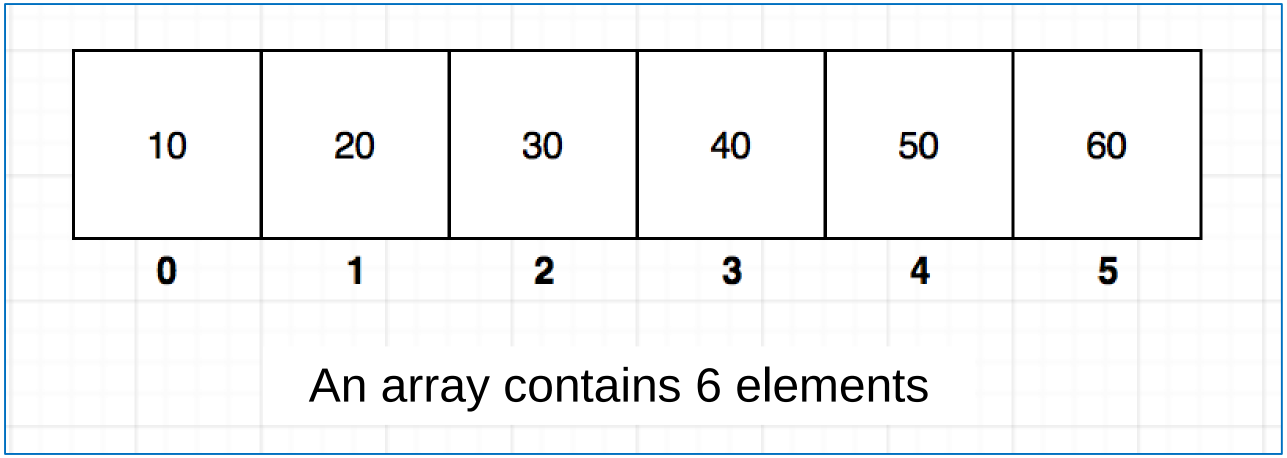
Why do we need array ?

- Problem: Student Management System
- Solution
  - (1) Group all data of each student together => use struct
    (2) Store N students => use array
    - Or we can use linked list

- C++
  - Use (array) to store contiguous elements with same data type
  - Use (pointer) to develop linked list if necessary

What is array ?

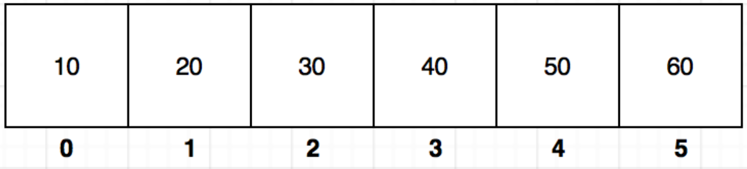
- Array is a list of element with same data type and allocated contiguously in memory.
- For example



10	20	30	40	50	60
0	1	2	3	4	5

An array contains 6 elements

What is array ?

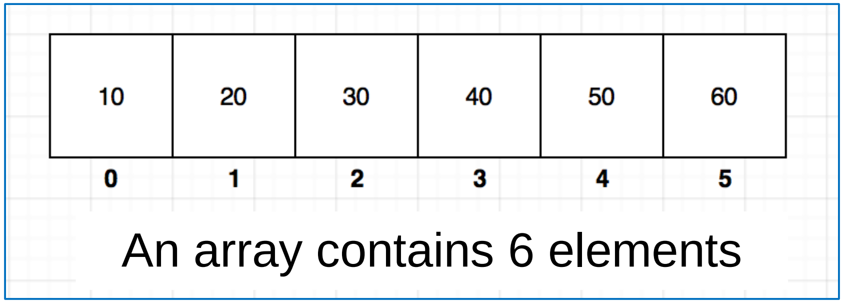


10	20	30	40	50	60
0	1	2	3	4	5

An array contains 6 elements

- An array contain 6 numbers
  - These numbers allocated contiguously in memory
  - So,
    - If the value of the first element is 10 and starts at the **100th BYTE** in the memory of the program
    - Then
      - Address of the second element: **104**
      - Address of the third element: **108**
      - Address of the fourth element: **112**
      - Address of the fifth element **116**
      - Address of the sixth element: **120**

What is array ?

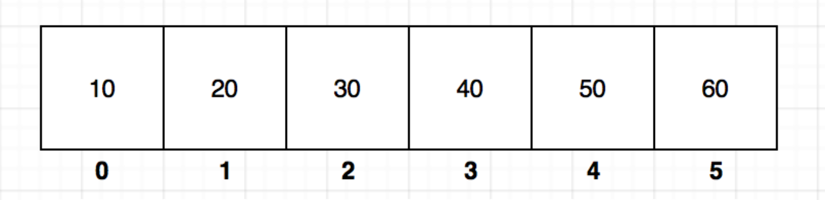


10	20	30	40	50	60
0	1	2	3	4	5

An array contains 6 elements

- An array contain 6 numbers
  - These numbers allocated contiguously in memory
  - These elements have index to access
    - The index of the first element is ALWAYS 0
    - The index of the second element is 1, and so on.
  - Therefore,
    - The index of memory cell containing value 10 is **0**
    - The index of memory cell containing value 20 is **1**
    - The index of memory cell containing value 30 is **2**
    - The index of memory cell containing value 40 is **3**
    - The index of memory cell containing value 50 is **4**
    - The index of memory cell containing value 60 is **5**

What is array ?

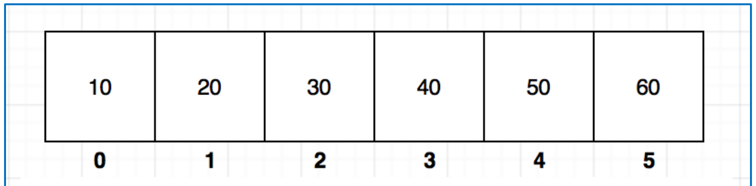


10	20	30	40	50	60
0	1	2	3	4	5

An array contains 6 elements

- An array contains 6 numbers
  - These numbers allocated contiguously in memory
  - These elements have index to access
    - The index of the first element is ALWAYS 0
    - The index of the second element is 1, and so on
  - Therefore,
    - If an array has **N** elements then the index of the last element will be (**N-1**) - not **N**

What is array ?

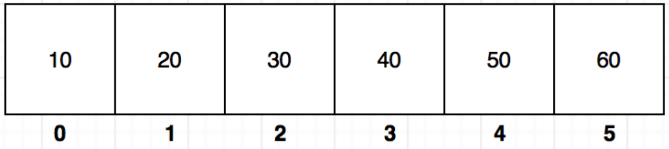


10	20	30	40	50	60
0	1	2	3	4	5

An array contains 6 elements

- To calculate the address of a memory cell with index k, the program uses the following formula:
  - Address = address of first element + k * (size of element)
  - Therefore, the program easily points out an element at any index => RANDOM ACCESS

What is array ?



10	20	30	40	50	60
0	**1**	**2**	**3**	**4**	**5**

An array contains 6 elements

- However, compiler must know the size of the array
  - Therefore, we can calculate address of k^{th} element in the memory by using this formula:

Address of k^{th} element = **first + k**

An array contains 6 elements

- **first**: address of the first element
  - **first** is name of variable of array

1D Array

Declaration

```
int main(){
int a[6];
int b[6] = {10, 20, 30};
int c[6] = {10, 20, 30, 40, 50, 60};

return 0;
}
```

- a: an array of 6 integers
  - Element values are unknown
- b: an array of 6 integers
  - First 3 element values are 10, 20, and 30
  - Last 3 element values are unknown
- c: an array of 6 integers
  - Element values are 10, 20, 30, 40, 50, and 60

1D Array

Declaration

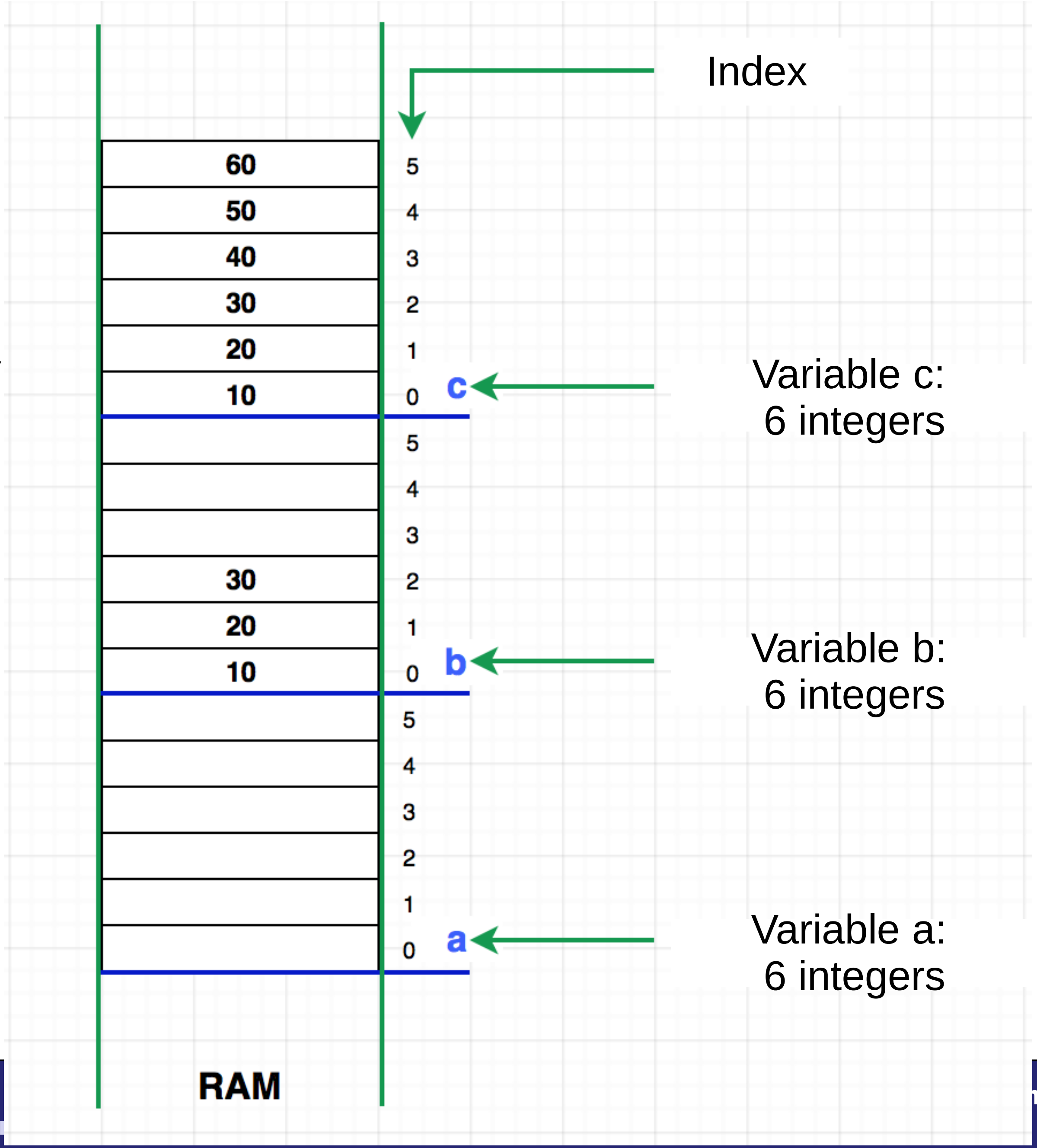
```
int main(){
int a[6];
int b[6] = {10, 20, 30};
int c[6] = {10, 20, 30, 40, 50, 60};

return 0;
}
```

- Notes:
    - All variables a, b, and c are array containing 6 elements
    - So, **the index starts from 0 to 5**

1D Array

Declaration

How are arrays (a, b, and c) allocated in memory?



1D Array
Declaration

- Number of elements in array
  - Must be determined at compile time
  - Is constant and non-negative
    - Use macro
      - #define MAX_SIZE
    - Use int const
      - const int max_size

```
#define MAX_SIZE 6

int main(){
const int max_size = 10;
int a[MAX_SIZE];
int b[max_size];
return 0;
}
```

1D Array

Read and write elements of 1D array

- Two ways:
  - By index
  - By address of memory

1D Array

Read and write element of 1D array

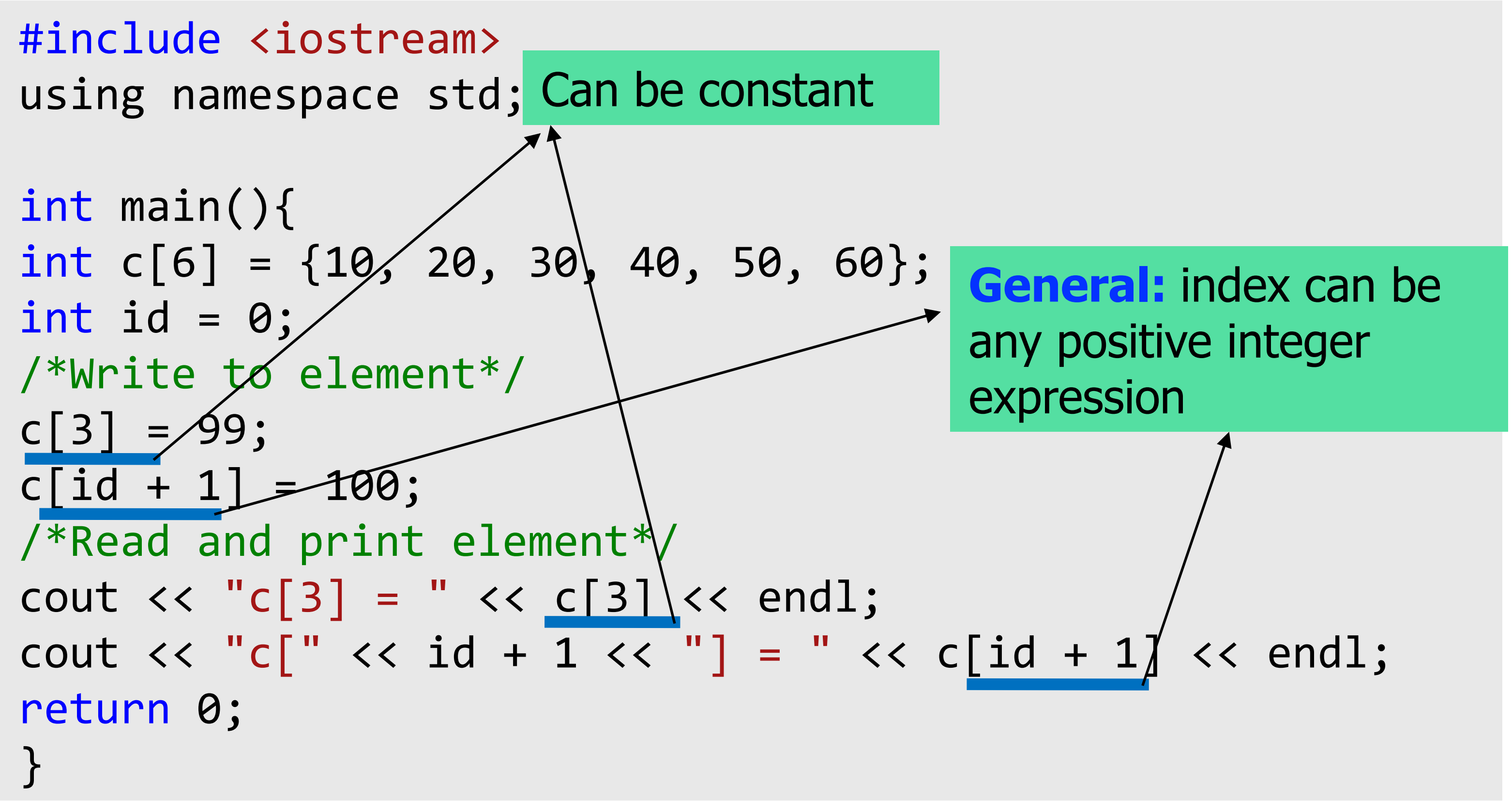
- Two ways:
  - By index

```
#include <iostream>
using namespace std;
int main(){
int c[6] = {10, 20, 30, 40, 50, 60};
int id = 0;
/*Write to element*/
c[3] = 99;
c[id + 1] = 100;
/*Read and print element*/
cout << "c[3] = " << c[3] << endl;
cout << "c[" << id + 1 << "] = " << c[id + 1] << endl;
return 0;
}
```

1D Array

Read and write element of 1D array

- Two ways:
  - By index



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

int main(){
int c[6] = {10, 20, 30, 40, 50, 60};
int id = 0;
/*Write to element*/
c[3] = 99;
c[id + 1] = 100;
/*Read and print element*/
cout << "c[3] = " << c[3] << endl;
cout << "c[" << id + 1 << "] = " << c[id + 1] << endl;
return 0;
}
```

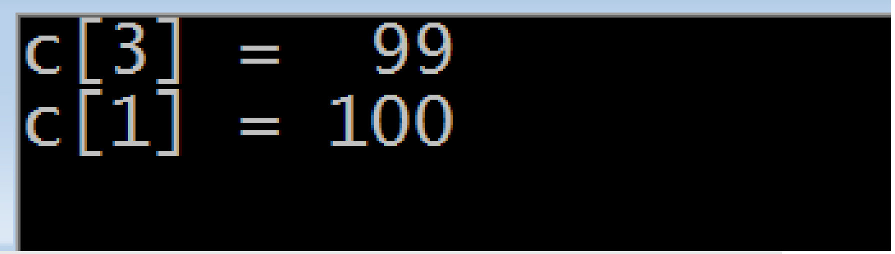
Can be constant

General: index can be any positive integer expression

1D Array

Read and write element of 1D array

- Two ways:
  - By index



```
c[3] =  99
c[1] = 100
```

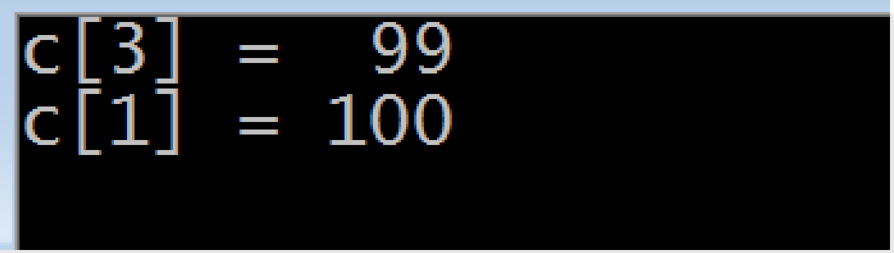
```
#include <iostream>
using namespace std;
int main(){
int c[6] = {10, 20, 30, 40, 50, 60};
int id = 0;
/*Write to element*/
c[3] = 99;
c[id + 1] = 100;
/*Read and print element*/
cout << "c[3] = " << c[3] << endl;
cout << "c[" << id + 1 << "] = " << c[id + 1] << endl;
return 0;
}
```

1D Array

Read and write element of 1D array

- Two ways:
  - By address in memory

```
#include <iostream>
using namespace std;
int main(){
int c[6] = {10, 20, 30, 40, 50, 60};
int id = 0;
/*Write to element*/
*(c + 3) = 99;
*(c + (id + 1)) = 100;
/*Read and print element*/
cout << "c[3] = " << *(c + 3) << endl;
cout << "c[" << id + 1 << "] = " << *(c + (id + 1)) << endl;
return 0;
}
```

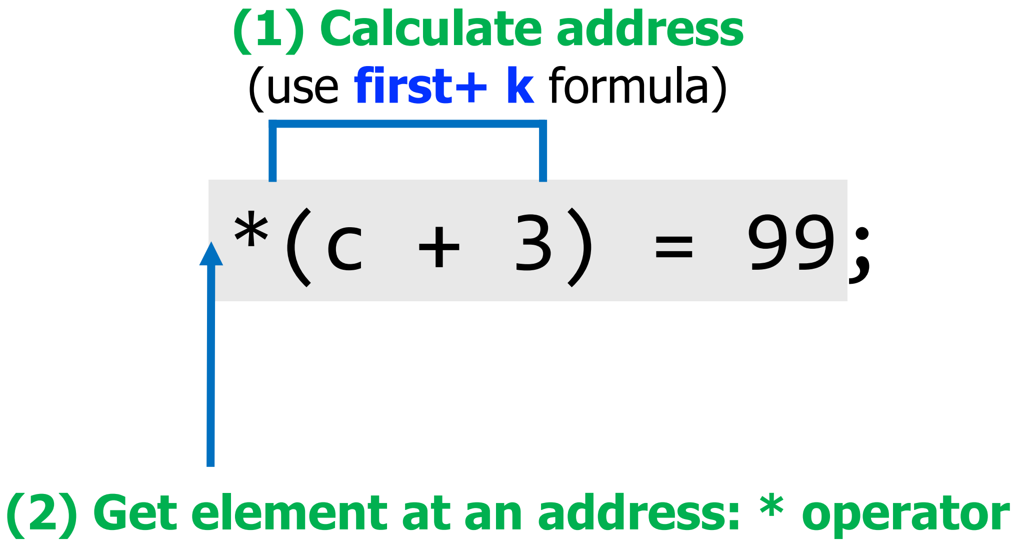


```
c[3] =  99
c[1] = 100
```

1D Array

Read and write element of 1D array

- Two ways:
  - By address in memory
    - (1) Calculate address
    - (2) Get element by calculated address



(1) Calculate address

(use **first+ k** formula)

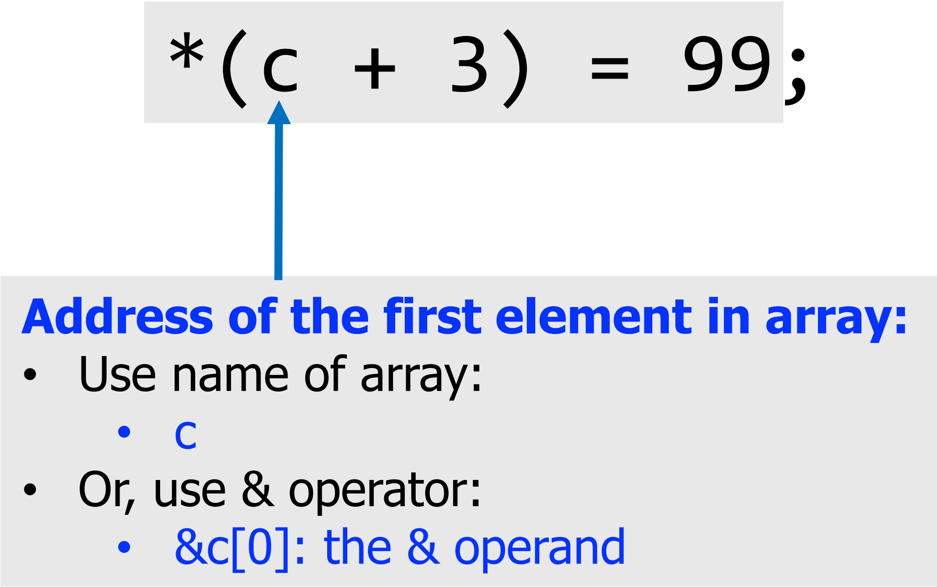
```
*(c + 3) = 99;
```

(2) Get element at an address: * operator

1D Array

Read and write element of 1D array

- Two ways:
  - Calculate the address and receive reference to the wanted element
    - Calculate the address
    - Receive reference to the wanted element



```
*(c + 3) = 99;
```

Address of the first element in array:

- Use name of array:
  - c
- Or, use & operator:
  - &c[0]: the & operand

1D Array

Some techniques in 1D array

- Access array elements
- Calculate statistical values from array
  - Sum
  - Maximum
  - Minimum
  - Median
  - Standard variation
  - Mean
  - ...
- Element-wise operation
  - Normalize all element (student, product, etc) in array
- Swap two elements in an array
  - Sorting
- Sort all elements in an array
- Find an element in an array
  - Binary search

1D Array

Some techniques in 1D array

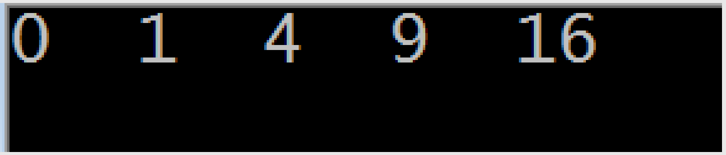
- Access array elements
  - Use 1 index variable (int type)
  - Assign value 0 to this variable
    - Indicate the first element of array
  - Loop through an array
    - For each iteration,
      - Access element by index: read or write
      - Increase index variable by 1

1D Array

Some techniques in 1D array

- Access array elements

```
#include <iostream>
using namespace std;
#define MAX_SIZE 100
int main(){
int arr[MAX_SIZE];
int cur_size = 5; //use 5 items only
/*Initialize array*/
for(int i=0; i<cur_size; i++){
    arr[i] = i*i;
}
/*Print array*/
for(int i=0; i<cur_size; i++){
    cout << arr[i] << “ “;
}
return 0;
}
```



```
0   1   4   9   16
```

1D Array

Some techniques in 1D array

- Access array elements

```
#include <iostream>
using namespace std;
#define MAX_SIZE 100
int main(){
int arr[MAX_SIZE];
int cur_size = 5; //use 5 items only
/*Initialize array*/
for(int i=0; i<cur_size; i++){
    arr[i] = i*i;
}
/*Print array*/
for(int i=0; i<cur_size; i++){
    cout << arr[i] << " ";
}
return 0;
}
```

MAX_SIZE (100) is positive integer

cur_sizee: number of elements are being used (can be determined by user)

For loop: iterate through all elements to read and print into console

1D Array
Some techniques in 1D array

- Access array elements
  - Exercise
    - Use for loop but the stopping condition should be put inside the for scope { }
      - For(;;){…}
      - Use break
    - Other loop types
      - while
      - do ... while

1D Array

Some techniques in 1D array

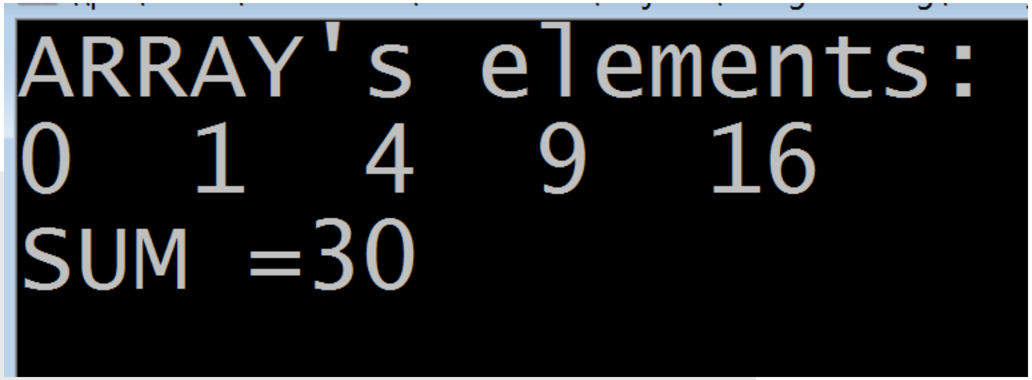
- Calculate sum of all elements in array
  - Loop
  - Recursion (will be covered in a future chapter)

1D Array

Some techniques in 1D array

- Calculate sum of all elements in array
    - Loop
        - Let sum be the sum of all elements in array
        - Initialize sum = 0
        - Use loop to iterate through all elements in array
            - For each iteration,
                - Read element by index
                - Add value the element at the specific index to sum
                - Increase index variable by 1

```
#include <iostream>
using namespace std;
#define MAX_SIZE 100

int main(){
int arr[MAX_SIZE];
int cur_size = 5; //use 5 items only
/*Initialize array*/
for(int i=0; i<cur_size; i++){
    arr[i] = i*i;
}
/*Print array*/
cout << "ARRAY's elements: " << endl;
for(int i=0; i<cur_size; i++){
    cout << arr[i] << “ ”;
}
//...
}
```



```
ARRAY's elements:
0   1   4   9   16
SUM =30
```

```
#include <iostream>
using namespace std;
#define MAX_SIZE 100
int main(){
int arr[MAX_SIZE];
int cur_size = 5; //use 5 items only
/*Initialize array*/
for(int i=0; i<cur_size; i++){
    arr[i] = i*i;
}
//...
/*Calculate sum*/
int sum = 0;
for(int i=0; i<cur_size; i++){
    sum += arr[i];
}
cout << "SUM = " << sum << endl;
return 0;
}
```

For loop: loop through all elements and add value to sum.

1D Array

Some techniques in 1D array

- Find maximum value
  - Let max_value be the maximum value
  - Initialize max_value = smaller than the smallest value
    - Or assign value of the first element of the array to max_value
  - Loop through all elements
    - For each element at index ID,
      - If value of this element LARGER THAN max_value
        - Assign max_value = value of this element
        - Increase index variable by 1

1D Array

Some techniques in 1D array

- Find minimum value
  - Let min_value be the minimum value
  - Initialize min_value = larger than the largest element
    - Or assign value of the first element in the array to min_value
  - Loop through all element
    - For each element at index ID,
      - If value of this element SMALLER THAN min_value
        - Assign min_value = value of this element
        - Increase index variable by 1

1D Array

Some techniques in 1D array

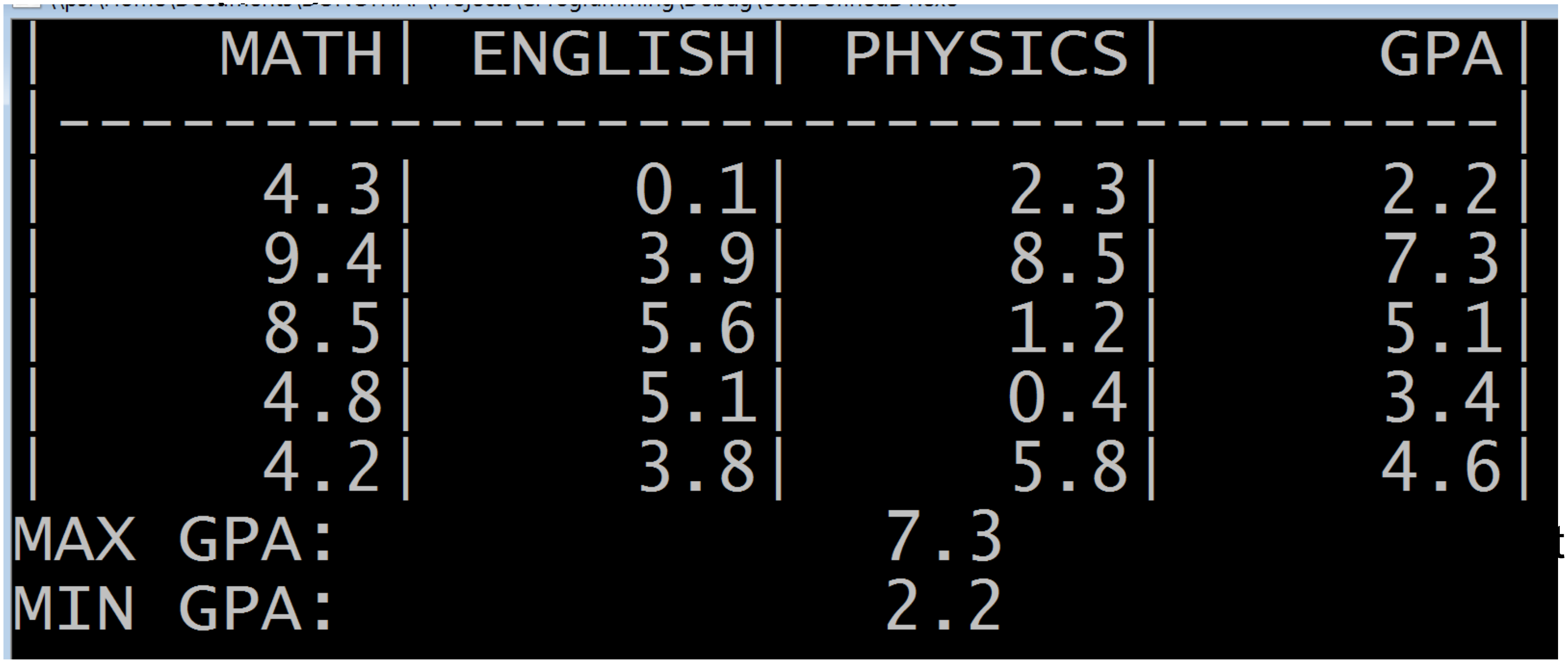
- Find maximum/minimum value
    - Problem
        - Each student has attributes:
            - Identifier (code), name (name), math score (math), english score (english), and phisics score (physics)
        - Let N be the number of students
        - Program starts with all score values assigned randomly from 0 to 10. Identifer and name of student do not need initialization

        - Find maximum and minimum value and print into console.

1D Array

Some techniques in 1D array

- Find maximum/minimum value



```
|      MATH|  ENGLISH|  PHYSICS|       GPA|
|------------------------------------------|
|       4.3|      0.1|      2.3|       2.2|
|       9.4|      3.9|      8.5|       7.3|
|       8.5|      5.6|      1.2|       5.1|
|       4.8|      5.1|      0.4|       3.4|
|       4.2|      3.8|      5.8|       4.6|
MAX GPA:               7.3
MIN GPA:               2.2
```

1D Array

Some techniques in 1D array

- Find maximum/minimum value
  - Analysis:
    - Need to define a new data type, Student, which contains the information fields as mentioned previously
    - Save a list of up to NUM_STUDENT students.
    - Initialize array as required
      - 3 score columns are random initialized from 0 to 10
    - Find highest and lowest average scores and print them on the console
      - Average = (math + english + physics) / 3
  - Implement the program

1D Array

Some techniques in 1D array

- Find maximum/minimum value
  - Struct (Student)

```
typedef struct sStudent{
char student_code[10];
char student_name[50];
float math, english, physics;
} Student;
```

1D Array

Some techniques in 1D array

- Find maximum / minimum value
  - Declare an array has NUM_STUDENT elements

```
#include <time.h>
#define NUM_STUDENT  5
typedef struct sStudent{
char student_code[10];
char student_name[50];
float math, english, physics;
} Student;

int main(){
/*List of students*/
Student list[MAX_SIZE];
/...
```

To use **time** function

1D Array

Some techniques in 1D array

- Find maximum/minimum value
  - Initialize array

```
/*Initialize the list*/
time_t t;
srand((unsigned) time(&t));
for(int i=0; i<NUM_STUDENT ; i++){
list[i].math = ((float)rand() / RAND_MAX)*10;
list[i].english = ((float)rand() / RAND_MAX)*10;
list[i].physics = ((float)rand() / RAND_MAX)*10;
}
```

Use **rand()** function to generate integers from 0 to **RAND_MAX** (constant)
rand()/RAND_MAX: from 0 to 1
(rand()/RAND_MAX)*10: from 0 to 10
srand: create random generator based on system time (function **time**). Without this, every time you run the program, the randomized values always stay the same.

1D Array

Some techniques in 1D array

- Find maximum / minimum value
  - Find largest / smallest element in array

```c
/*Find max gpa and min gpa*/
float gpa_max = -1.0f;
float gpa_min = 11.0f;
float gpa;
for(int i=0; i<NUM_STUDENT; i++){
gpa = (list[i].math + list[i].english + list[i].physics)/3;
if(gpa_max < gpa) gpa_max = gpa;
if(gpa_min > gpa) gpa_min = gpa;
}
```

1D Array

Some techniques in 1D array

- Find maximum / minimum value
  - Print array and largest, smallest element into console

```
/*Print scoreboard, max gpa, and min gpa*/
cout << "|" << setw(8) << "MATH"
        << "|" << setw(8) << "ENGLISH"
        << "|" << setw(8) << "PHYSICS"
        << "|" << setw(8) << "GPA|" << endl;
cout << "|-----------------------------------|" << endl;
for(int i=0; i<NUM_STUDENT; i++){
gpa = (list[i].math + list[i].english + list[i].physics)/3;
cout << "|" << setw(8) << list[i].math
              << "|" << setw(8) << list[i].english
              << "|" << setw(8) << list[i].physics
              << "|" << setw(8) << gpa << endl;
}
cout << "MAX GPA:" << gpa_max << endl;
cout << "MIN GPA:" << gpa_min << endl;
```

2D array
Application

- Matrices in mathematics (Linear algebra) are 2D arrays
- Digital image is a 2D array of pixels
- Graph (network of objects) can be represented using 2D arrays

2D array

Model vs Physical storage

10	20	30	40
50	60	70	80
90	100	110	120

A model of an 2D array has: **3 rows x 4 columns**

10	20	30	40	50	60	70	80	90	100	110	120

Physical storage of 2D array: linearize 2D array
Method: array is stored row after row

2D array

How are 2D arrays stored?

- Elements are stored consecutively, row after row
- If the first element (value 10) begins at BYTE with address 100
  - Element with value 20 has address: **104**
  - Element with value 30 has address: **108**
  - Element with value 50 has address: **116**
  - Element with value 60 has address: **120**
  - Element with value 90 has address: **132**
  - Element with value 100 has address: **136**
  - V.v

2D array

How are 2D arrays stored?

- Elements are saved consecutively, row after row
- Elements in 2D array are indexed for accessing, using 2 types of index
  - Let **row** and **col** be indexes of an element
  - Row and col range from 0 to (numRow -1) and (numCol -1) respectively.

col

		0	**1**	**2**	**3**
	0	10	20	30	40
row	**1**	50	60	70	80
	2	90	100	110	120

2D array

How are 2D arrays stored?

- Elements are saved consecutively, row after row
- Elements in 2D array are indexed for accessing, using 2 types of index.
- Program can calculate the address of the start memory block of element [row, col] easily
  - Address of element [row, col] =
    address of the first element +
    [row* (number of elements on each row) +
    col] * size of element

				col	
		0	1	2	3
	0	10	20	30	40
row	1	50	60	70	80
	2	90	100	110	120

2D array

How are 2D arrays stored?

- C++ compiler knows the size of an element. Therefore, programmers can calculate the address of the element [row, col]

Address of element **[row, col]** =
first + [row* **COLS** + **col]**

first: address of the first element

- Which is the array name

COLS: number of elements on each row

col

		0	**1**	**2**	**3**
	0	10	20	30	40
row	**1**	50	60	70	80
	2	90	100	110	120

2D array

2D array declaration

```
int main(){
int a[3][4];
int b[3][4] = {   {10, 20, 30} };
int c[3][4] = {   {10, 20, 30, 40},
           {50, 60},
       };
int d[3][4] = {   {10, 20, 30, 40},
           {50, 60, 70, 80},
           {90, 100, 110, 120}
       };
return 0;
}
```

2D array

2D array declaration

```
int main(){
int a[3][4];
int b[3][4] = {   {10, 20, 30} };
int c[3][4] = {   {10, 20, 30, 40},
               {50, 60},
          };
int d[3][4] = {   {10, 20, 30, 40},
               {50, 60, 70, 80},
               {90, 100, 110, 120}
          };
return 0;
}
```

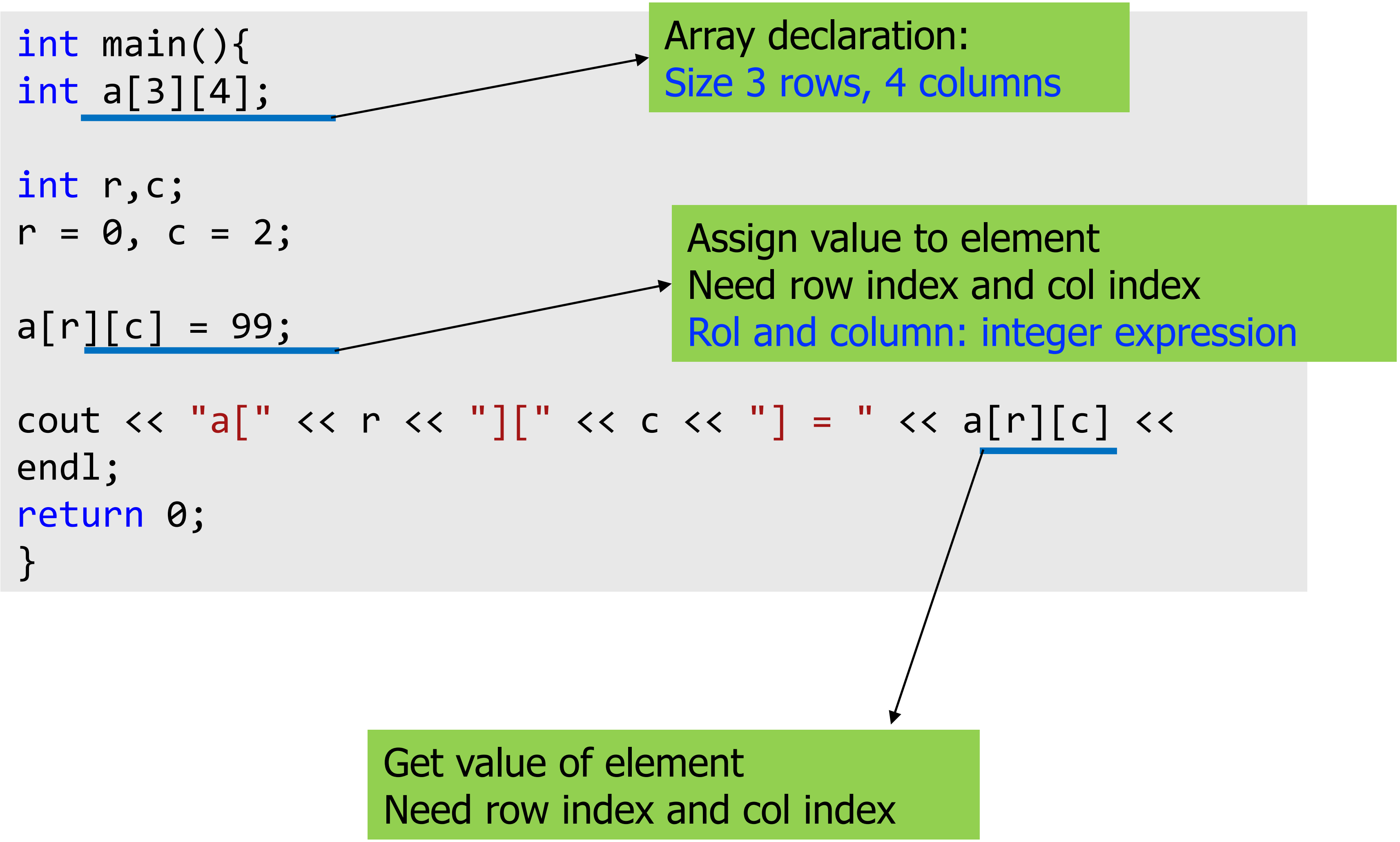
Declare a 2D array without initialization
Size: 3 rows, 4 columns

Declared with incompletely initialization
Size: 3 rows, 4 columns

Declared with complete initialization
Size: 3 rows, 4 columns

2D array

Read and write elements of 2D array



```
int main(){
int a[3][4];

int r,c;
r = 0, c = 2;

a[r][c] = 99;

cout << "a[" << r << "][" << c << "] = " << a[r][c] <<
endl;
return 0;
}
```

Array declaration:
Size 3 rows, 4 columns

Assign value to element
Need row index and col index
Rol and column: integer expression

Get value of element
Need row index and col index

2D array

Some techniques in 2D array

- Access array elements
- Access elements in the same row
- Access elements in the same column
- In square matrix
  - Access elements in the main diagonal
  - Access elements in the secondary diagonal

  - Access elements above the main diagonal
  - Access elements below the main diagonal

2D array
Access array elements

- Let ROWS and COLS be the total number of rows and columns respectively
  - ROWS and COLS are constants
    - Through #define
    - Through const int ROWS, COLS;
- Let row and col be two variables indicating the row index and column index
  - row: row index
  - col: column index

2D array

Access array elements

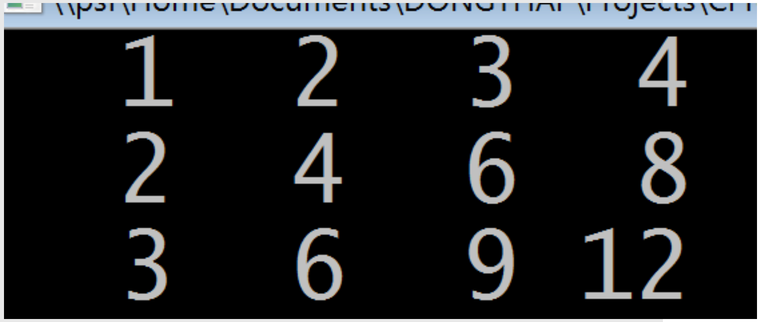
- Let ROWS and COLS be the total number of rows and columns respectively
  - ROWS and COLS are constants
    - Through #define
    - Through const int ROWS, COLS;
- Let row and col be two variables indicating the row index and column index
  - row: row index
  - col: column index
- Use to nested loop
  - For each row
    - For each column
      - Retrieve element at [row, col] to write or read
      - Increase the column index (col) to access the next element in the same row
    - Increase the row index (row) to access the next row

2D array

Accessing array elements

- Note
  - According to this method, array elements are accessed sequentially in each row, from one row to another.
    - Column index increases faster than row index
  - This method is more effective than the column-based retrieving methods
    - Row index increases faster than column index

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

int main(){
const int ROWS = 3, COLS = 4;
int a[ROWS][COLS];
int row,col;
/*Initialize array*/
for(row=0; row<ROWS; row++){
    for(col=0; col<COLS; col++){
        a[row][col] = (row + 1)*(col + 1);
    }
}
/*Print array*/
for(row=0; row<ROWS; row++){
    for(col=0; col<COLS; col++){
        cout << a[row][col] << “ “;
    }
    cout << endl;
}
return 0;
}
```



```
   1   2   3   4
   2   4   6   8
   3   6   9  12
```

Loop over the rows, then the columns (nested)

Access and assign value

Access and print value

New line after printing each row

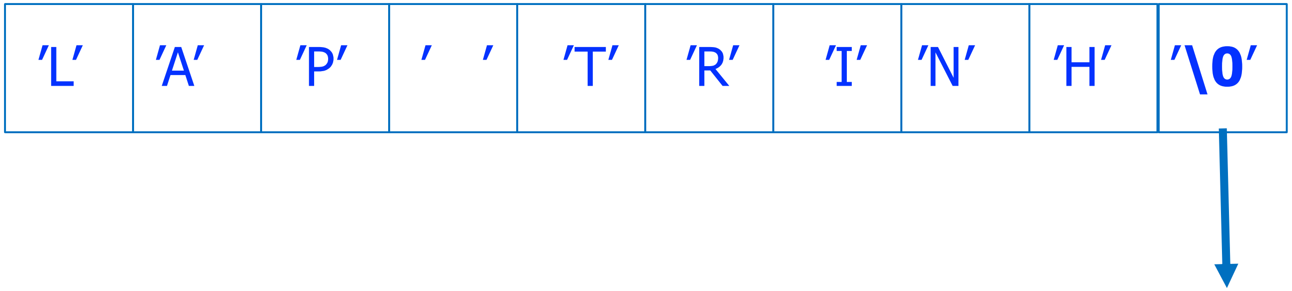
String

Content

- String in C++
- String declaration in C++
- String processing functions
  - Print string function
  - Read string function
  - Get length of string function
- Some techniques
  - Find substring
  - Remove whitespace between words and trailing whitespace
  - Concatenate strings
  - Split string
    - Into tokens
    - Into first name and surname

String in C++

- In C++, String is an array of characters which is terminated by a special null character ‘\0’
- => A character array of size N can hold only up to (N-1) characters
- Example: string “LAP TRINH”
  - Length: 9 characters
  - Number of necessary memory blocks: 10

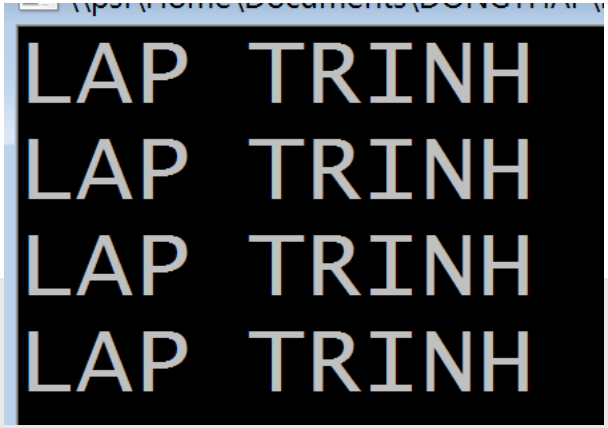


′L′	′A′	′P′	′ ′	′T′	′R′	′I′	′N′	′H′	**′\0′**

String ending with special character

String declaration

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

int main(){
const int MAX_LEN = 50;
char s1[MAX_LEN];
char s2[MAX_LEN] =
    {'L', 'A', 'P', ' ', 'T', 'R', 'I', 'N', 'H', '\0'};
char s3[MAX_LEN] = "LAP TRINH";
char s4[] =
    {'L', 'A', 'P', ' ', 'T', 'R', 'I', 'N', 'H', '\0'};
char s5[] = "LAP TRINH";

cout << s2 << endl << s3 << endl << s4 << endl << s5 << endl;
return 0;
}
```



```
LAP TRINH
LAP TRINH
LAP TRINH
LAP TRINH
```

String declaration

- `char s1[MAX_LEN];`
  - s1: can hold up to (MAX_LEN – 1) characters

- `char s2[MAX_LEN] =`

  `{'L', 'A', 'P', ' ', 'T', 'R', 'I', 'N', 'H', '\0'};`
  - s2: can hold up to (MAX_LEN – 1) characters
  - String initialization using array initialization → **need to be terminated by '\0'**

- `char s3[MAX_LEN] = "LAP TRINH";`
  - s3: can hold up to (MAX_LEN – 1) characters
  - Initialization using constant → **no need for '\0'**

String declaration

-
  ```
  char s4[] =
  {'L', 'A', 'P', ' ', 'T', 'R', 'I', 'N', 'H', '\0'};
  ```
  - s4: array of 10 memory blocks, hold exactly 9 characters of string “LAP TRINH”
  - No need to specify the array size when declaring with string initialization
  - Initialize the same way as array initialization

-
  ```
  char s5[] = "LAP TRINH";
  ```
  - s5: array of 10 memory blocks, hold exactly 9 of string “LAP TRINH”
  - No need to specify the array size when declaring with string initialization
  - Initialize using constant “LAP TRINH”

Some string function library

- Print string function
  - Function: cout

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

int main(){
const int MAX_LEN = 50;
char s1[MAX_LEN] =
    {'L', 'A', 'P', ' ', 'T', 'R', 'I', 'N', 'H', '\0'};
char s2[] = "LAP TRINH";

cout << s1 << endl << s2 << endl;
return 0;
}
```

Print our two strings, s1 and s2

Some string function library

- Read string function: read a word
  - Function: cin

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

int main(){
string str;
cout << "Enter a word: ";
cin  >> str;

cout << str << endl;
return 0;
}
```

cin: Read until reaching a whitespace → read word

Some string function library

- Read string function: read a line
  - Function: getline read until reaching a newline character (ENTER)

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

int main(){
string str;
cout << "Enter a line: ";
getline(cin, str);

cout << str << endl;
return 0;
}
```

getline: Read until reaching a newline character → read line

Some string function library

- Read string function: read a line
  - Use getchar(), until reaching newline character (ENTER)

```
#include <iostream>
#include <stdio.h>
using namespace std;

int main(){
    const int max_len = 50;
    char str[max_len], ch = '\0';
    int i=0;
    cout << "Enter a string, " << max_len << "chars max: " << endl;
    while(ch!='\n'){
        ch=getchar();
        str[i]=ch;
        i++;
    }
    str[i]='\0'; //ending string
    cout <<"line: " << str;
    return 0;
}
```

getchar: Read each character

Some string function library

- Read string function:
    - Should not use cin and getline in the same program
        - cin: Do not read newline character → use getline right after cin can return value immediately without input from user.

Some string function library

- Another functions:

Function	Explanation
strlen	Get length of a string
strcpy	Copy one string to another
strcmp	Compare two strings
strstr	Find string in string

Further reading: http://www.cplusplus.com/reference/cstring/