哈爾濱工業大學（深圳）
HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

高级语言程序设计
High-level Language Programming

Lecture 6 Arrays and Structures

Yitian Shao (shaoyitian@hit.edu.cn)
School of Computer Science and Technology

Arrays and Structures

Course Overview

- Arrays
  - Initialization
  - Two-dimensional arrays
  - Multi-dimensional arrays

- Structures
  - Declaration and initialization
  - Nested structures
  - Arrays of structures

- The typedef statement

- Enumerated data types

Try it yourself on https://www.onlinegdb.com

Arrays

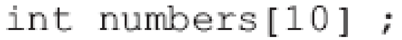
- A group of variables of the **same data type**

int a, b, c, d, e, …   **10 variables? 100?**

Arrays

- A group of variables of the **same data type**

- Define an array

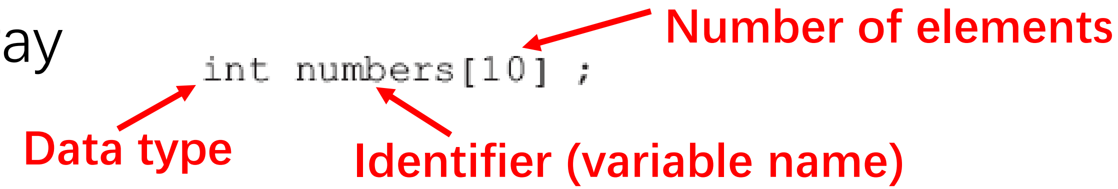


```
int numbers[10] ;
```

Arrays

- A group of variables of the **same data type**

- Define an array



```
int numbers[10] ;
```

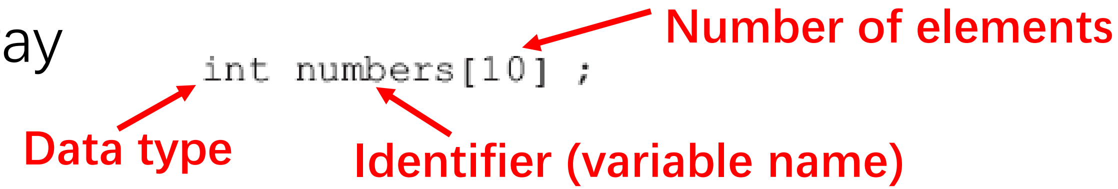
Number of elements

Data type

Identifier (variable name)

Arrays

- A group of variables of the **same data type**
- Define an array



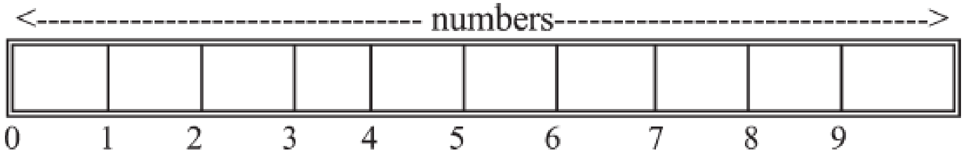
```
int numbers[10] ;
```

Number of elements

Data type

Identifier (variable name)

- Data storage

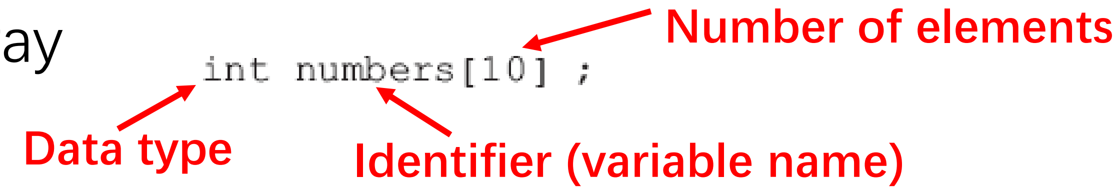


<------------------------------------ numbers------------------------------------>
0
1
2
3
4
5
6
7
8
9

Arrays

- A group of variables of the **same data type**

- Define an array



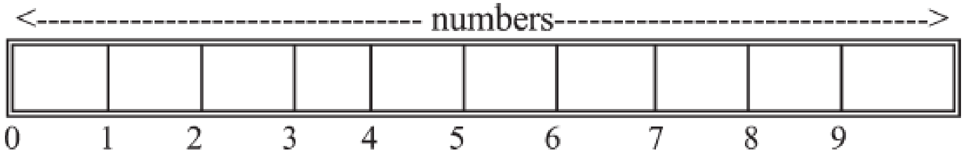
```
int numbers[10] ;
```

Number of elements

Data type

Identifier (variable name)

- Data storage



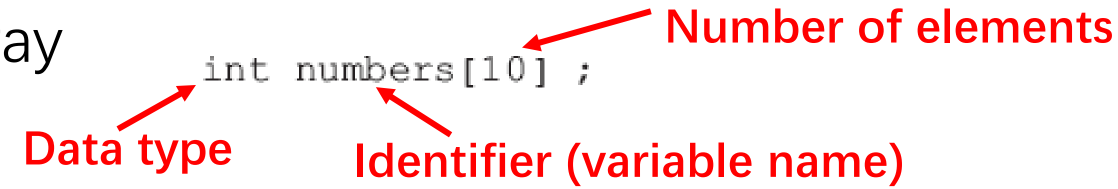
<------------------------------- numbers------------------------------->
0
1
2
3
4
5
6
7
8
9

- Size of the array: the **number of elements** in an array

Arrays

- A group of variables of the **same data type**

- Define an array



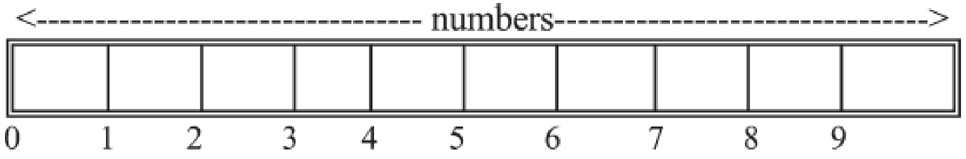
```
int numbers[10] ;
```

Number of elements

Data type

Identifier (variable name)

- Data storage

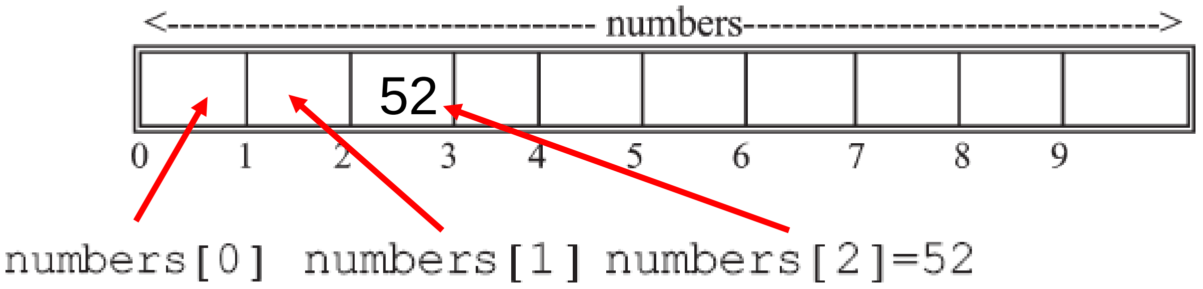


- Size of the array: the **number of elements** in an array

- Array index: The **position** of an element in an array
  Note: The index of the array numbers starts with 0

Arrays

- How to refer to a particular element ?
  - Use the **array name** and the **index** in brackets



numbers[0] numbers[1] numbers[2]=52

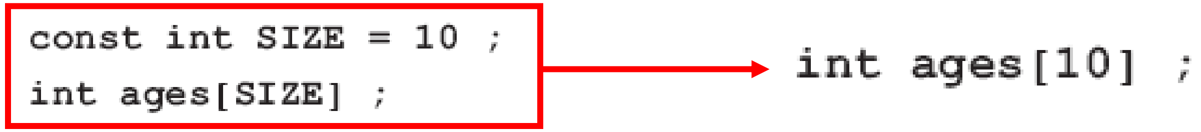
- How to access the final element?

numbers[9] √ numbers[10] ✗

Arrays

- Can use a const integer to specify the size of an array
  - The **const** keyword is used in the definition to specify that its value cannot be changed.
  - The identifier is usually written in uppercase.
  - Using a symbolic constant makes the program easier to modify

- Example: define a constant integer SIZE and an array “ages” with size SIZE



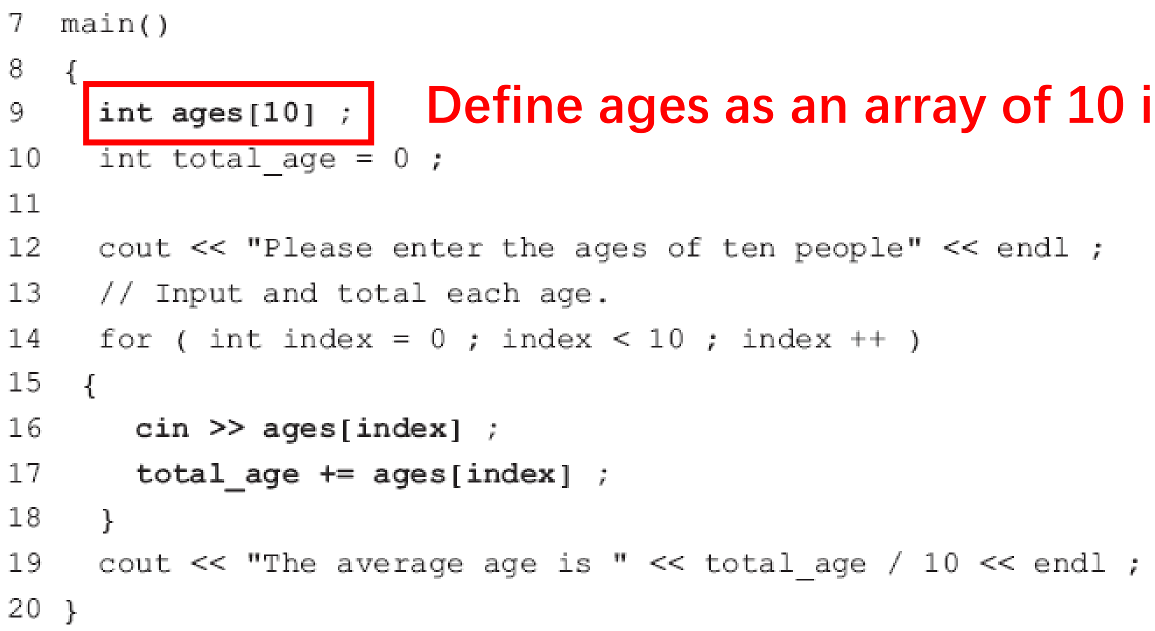
```
const int SIZE = 10 ;
int ages[SIZE] ;
```

```
int ages[10] ;
```

Arrays

Example: Compute the average age of 10 people

Recall:
data_type variable_name [number_of_elements]

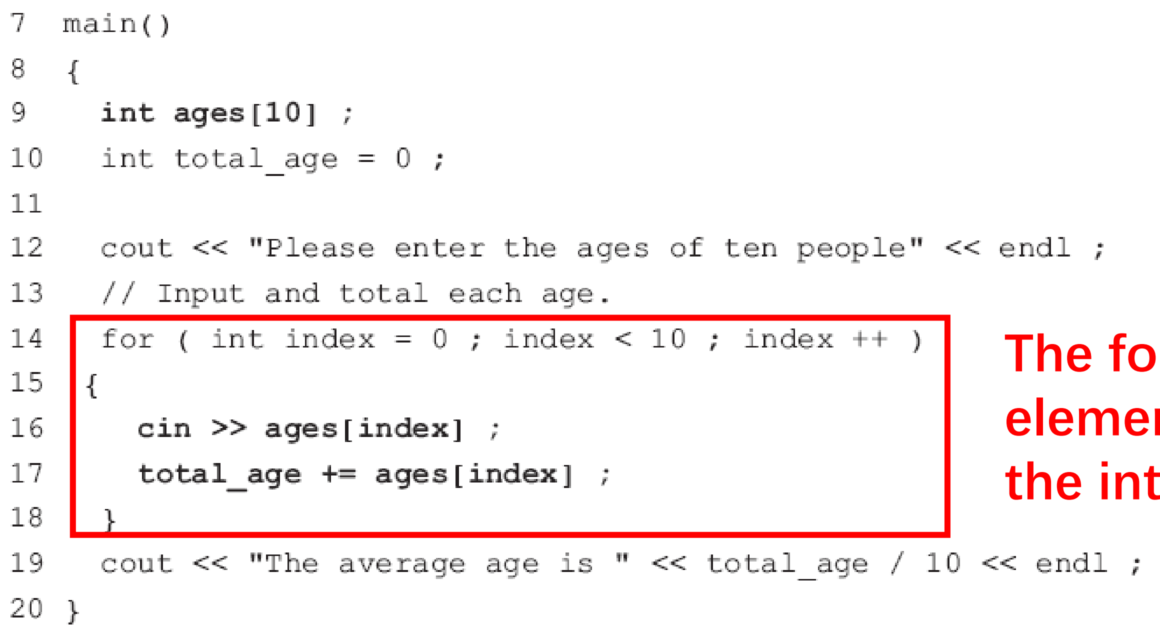


```
main()
{
  int ages[10] ;
  int total_age = 0 ;

  cout << "Please enter the ages of ten people" << endl ;
  // Input and total each age.
  for ( int index = 0 ; index < 10 ; index ++ )
 {
    cin >> ages[index] ;
    total_age += ages[index] ;
  }
  cout << "The average age is " << total_age / 10 << endl ;
}
```

Define ages as an array of 10 integers

Arrays

Example: Compute the average age of 10 people



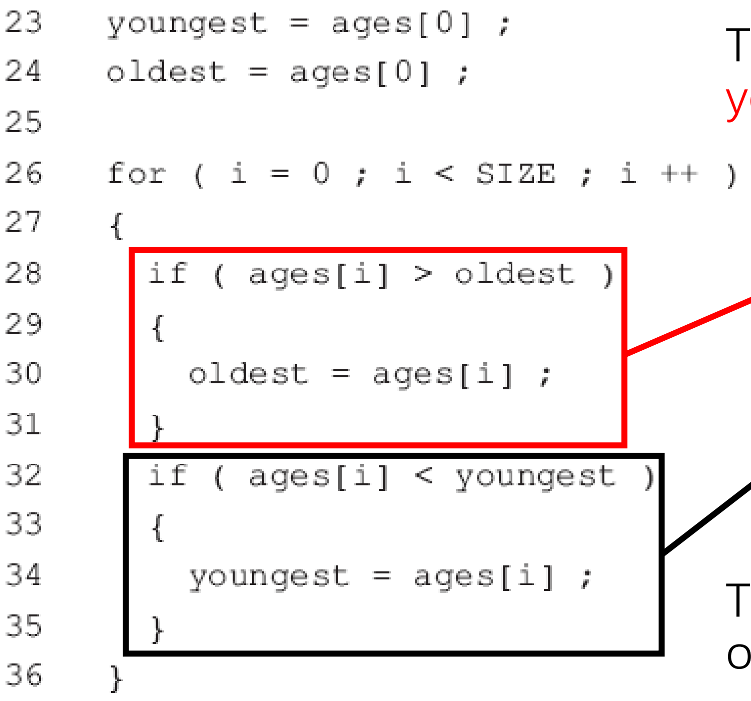
```
main()
{
 int ages[10] ;
 int total_age = 0 ;

 cout << "Please enter the ages of ten people" << endl ;
 // Input and total each age.
 for ( int index = 0 ; index < 10 ; index ++ )
 {
  cin >> ages[index] ;
  total_age += ages[index] ;
 }
 cout << "The average age is " << total_age / 10 << endl ;
}
```

The for loop is used to read each element of the array and add them to the integer variable “total_age”

Arrays

Example: Find the minimum and maximum values in an array

Assume we already stored the age of 10 people inside the array `int ages[10]`



```
youngest = ages[0] ;
oldest = ages[0] ;

for ( i = 0 ; i < SIZE ; i ++ )
{
  if ( ages[i] > oldest )
  {
    oldest = ages[i] ;
  }
  if ( ages[i] < youngest )
  {
    youngest = ages[i] ;
  }
}
```

The for loop compares each element in the array with the values youngest and oldest.

When ages[i] larger than oldest, its value is assigned to oldest.

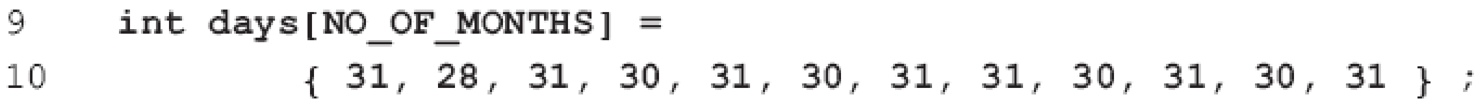
When ages[i] less than youngest , its value is assigned to youngest .

The smallest element of the array is in youngest and the largest is in oldest when the loop is completed.

Initialize an array

- Define and initialize an array with variable name “days”

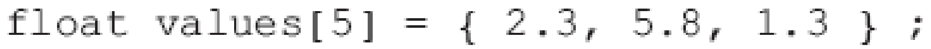
- The initial values in the array are separated by , and placed between { }



```
int days[NO_OF_MONTHS] =
           { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 } ;
```

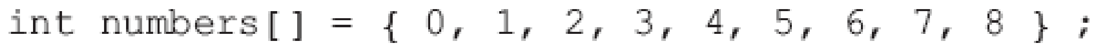
Initialize an array

- When the list of initial values is less than the number of elements in the array, the **remaining elements are initialized as 0**



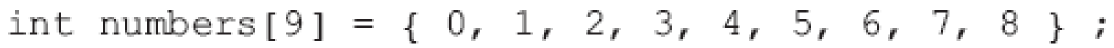
```
float values[5] = { 2.3, 5.8, 1.3 } ;
```

- If an array is defined **without specifying the number of elements** and is initialized to a series of values, the number of elements in the array is taken to be the same as the number of initial values.



```
int numbers[] = { 0, 1, 2, 3, 4, 5, 6, 7, 8 } ;
```

||

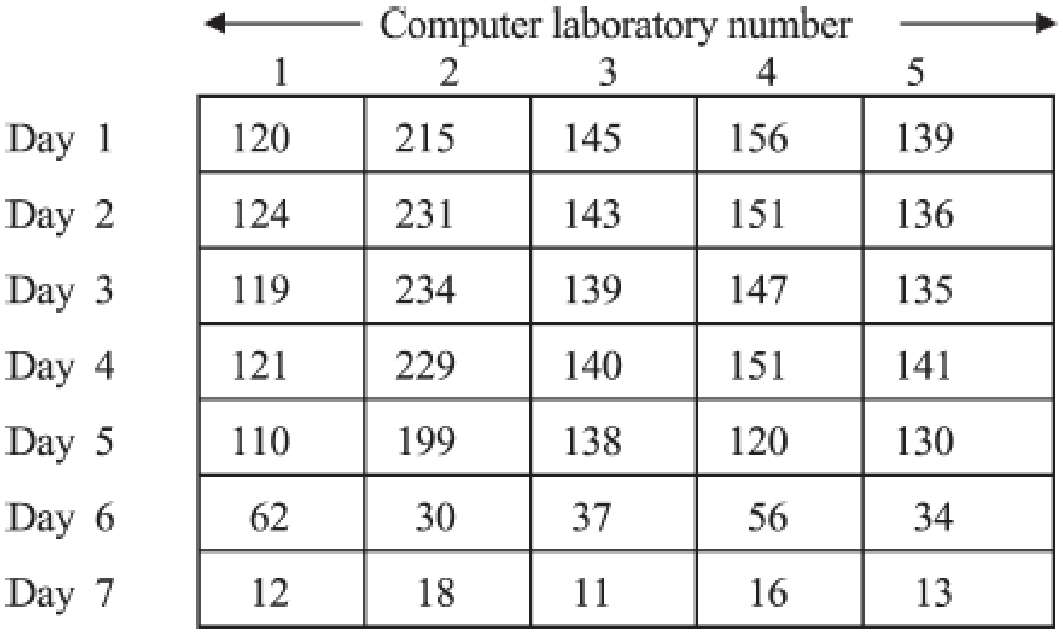


```
int numbers[9] = { 0, 1, 2, 3, 4, 5, 6, 7, 8 } ;
```

Two-dimensional arrays

- A two-dimensional array has more than one row of elements

Example: Number of usages of five labs for a week (7 days)



← Computer laboratory number →

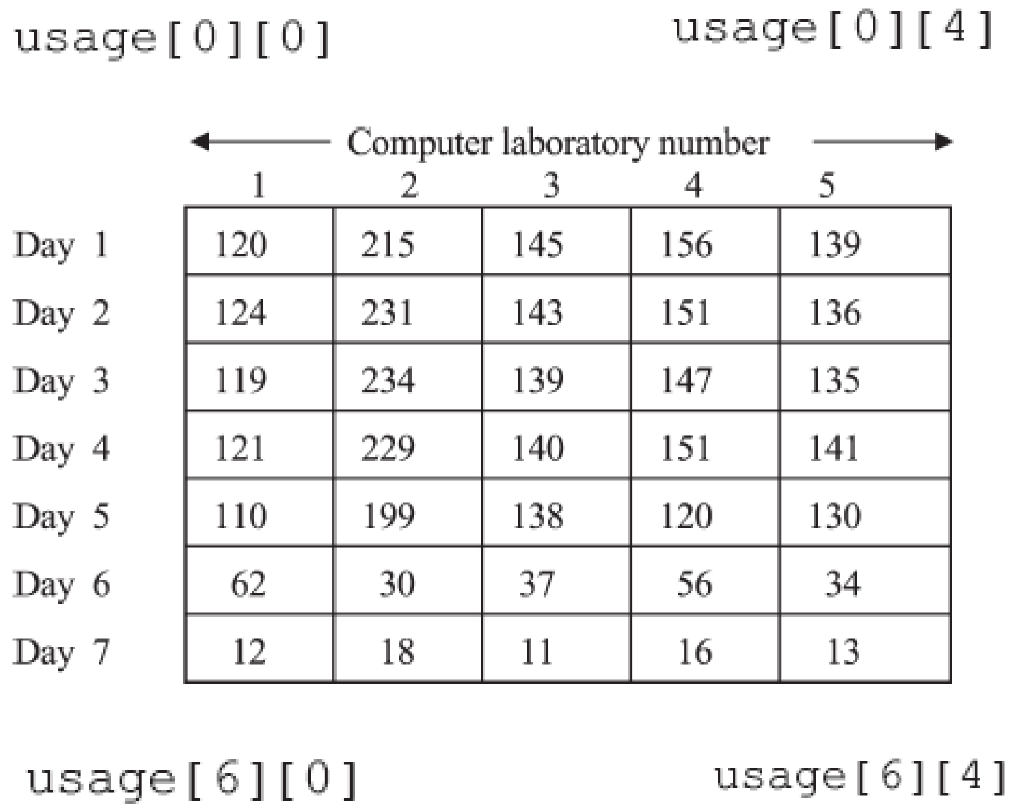
	1	2	3	4	5
Day 1	120	215	145	156	139
Day 2	124	231	143	151	136
Day 3	119	234	139	147	135
Day 4	121	229	140	151	141
Day 5	110	199	138	120	130
Day 6	62	30	37	56	34
Day 7	12	18	11	16	13

Two-dimensional arrays

- **Define**: enclose each dimension of the array in brackets

```
int usage[7][5] ;
```

- **Access an element**: specify the row and the column
  - **7 Rows** (days); the **row index** starts at 0 and ends at 6
  - **5 Columns** (labs); the **column index** starts at 0 and ends at 4



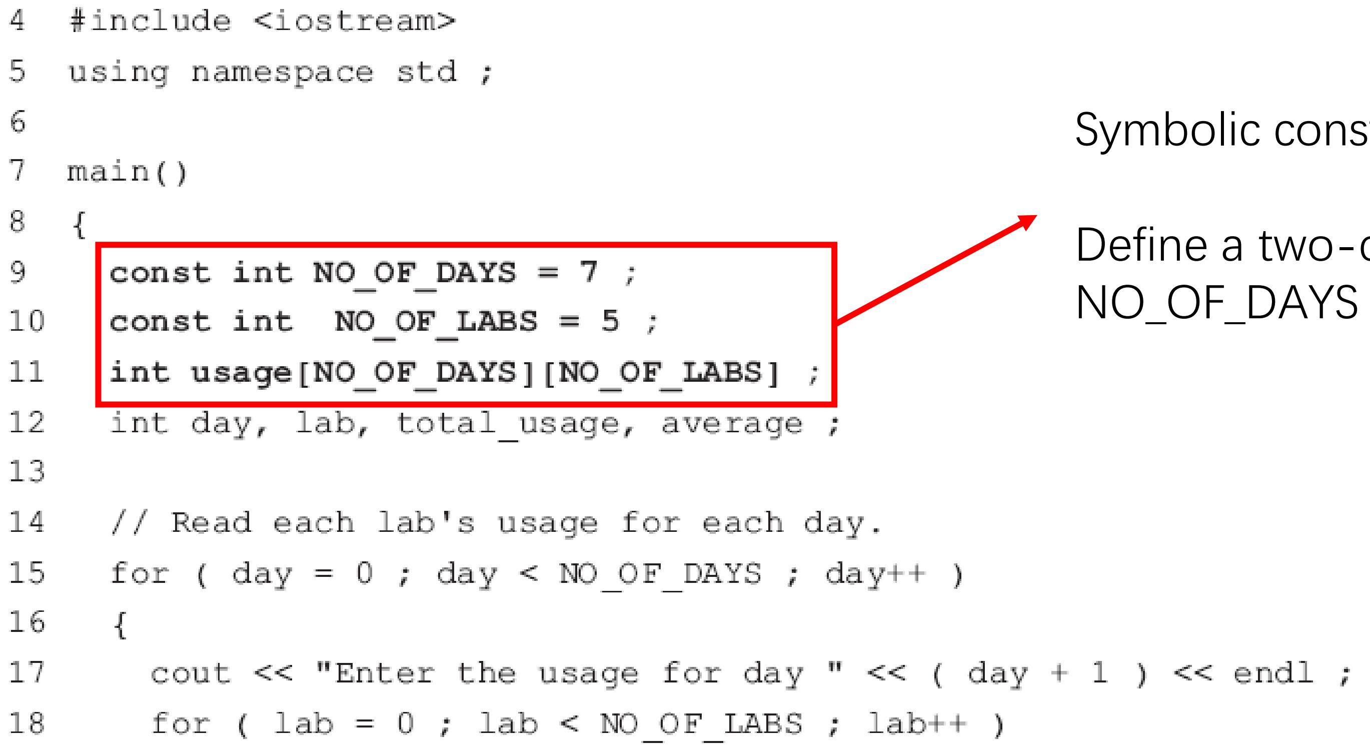
usage[0][0] usage[0][4]

	← Computer laboratory number →				
	1	2	3	4	5
Day 1	120	215	145	156	139
Day 2	124	231	143	151	136
Day 3	119	234	139	147	135
Day 4	121	229	140	151	141
Day 5	110	199	138	120	130
Day 6	62	30	37	56	34
Day 7	12	18	11	16	13

usage[6][0] usage[6][4]

Two-dimensional arrays

Example: reads in the number of students using the five laboratories over seven days



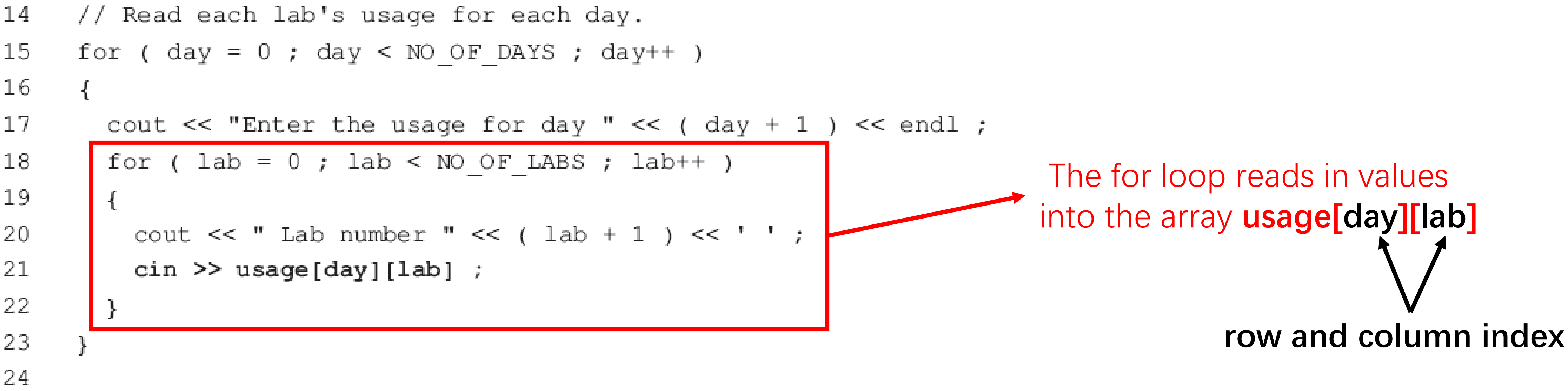
```
#include <iostream>
using namespace std ;

main()
{
  const int NO_OF_DAYS = 7 ;
  const int  NO_OF_LABS = 5 ;
  int usage[NO_OF_DAYS][NO_OF_LABS] ;
  int day, lab, total_usage, average ;

  // Read each lab's usage for each day.
  for ( day = 0 ; day < NO_OF_DAYS ; day++ )
  {
    cout << "Enter the usage for day " << ( day + 1 ) << endl ;
    for ( lab = 0 ; lab < NO_OF_LABS ; lab++ )
```

Symbolic constant NO_OF_DAYS and NO_OF_LABS.

Define a two-dimensional array usage with NO_OF_DAYS rows and NO_OF_LABS columns.

Two-dimensional arrays

Example: reads in the number of students using the five laboratories over seven days



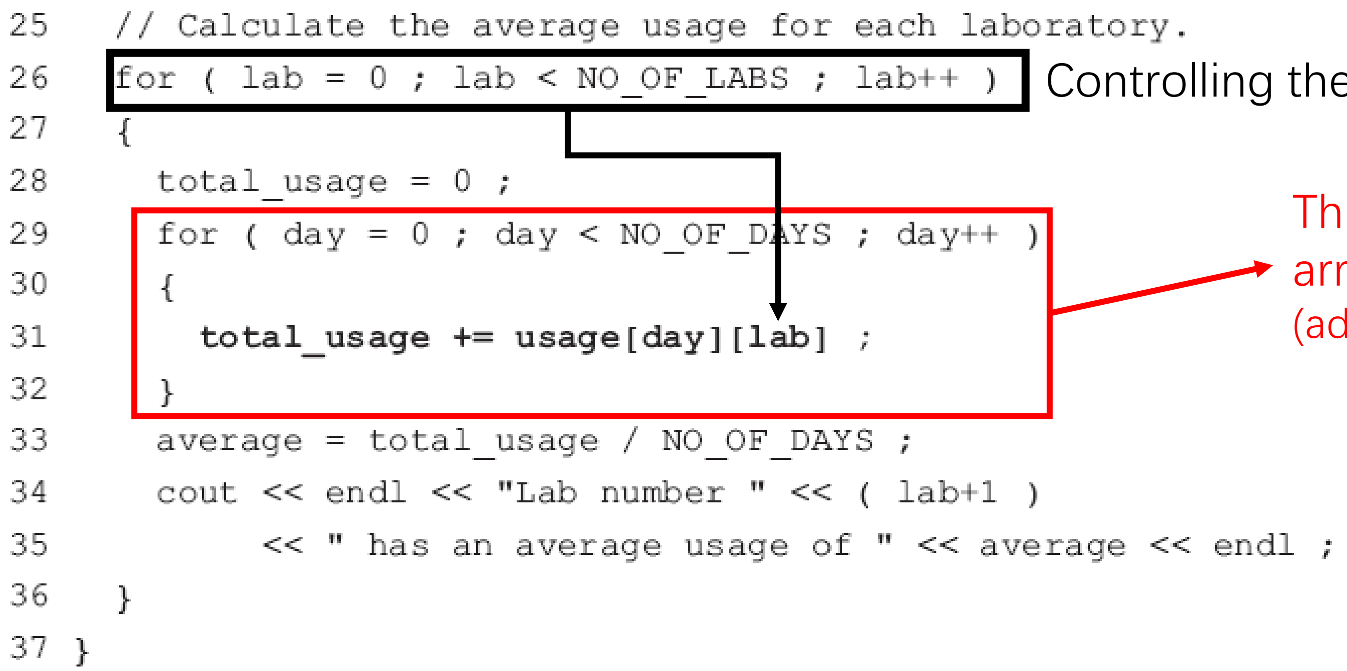
```
// Read each lab's usage for each day.
for ( day = 0 ; day < NO_OF_DAYS ; day++ )
{
  cout << "Enter the usage for day " << ( day + 1 ) << endl ;
  for ( lab = 0 ; lab < NO_OF_LABS ; lab++ )
  {
    cout << " Lab number " << ( lab + 1 ) << ' ' ;
    cin >> usage[day][lab] ;
  }
}
```

The for loop reads in values into the array **usage[day][lab]**

row and column index

Two-dimensional arrays

Example: reads in the number of students using the five laboratories over seven days



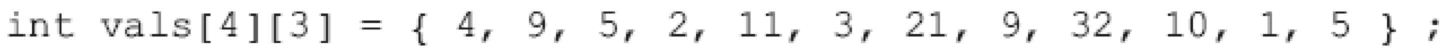
```
  // Calculate the average usage for each laboratory.
  for ( lab = 0 ; lab < NO_OF_LABS ; lab++ )
  {
    total_usage = 0 ;
    for ( day = 0 ; day < NO_OF_DAYS ; day++ )
    {
      total_usage += usage[day][lab] ;
    }
    average = total_usage / NO_OF_DAYS ;
    cout << endl << "Lab number " << ( lab+1 )
         << " has an average usage of " << average << endl ;
  }
}
```

Controlling the **column index** of the array **usage**

This for loop totals every **column** of the array **usage** and average them (adding elements by increasing the **row index**)

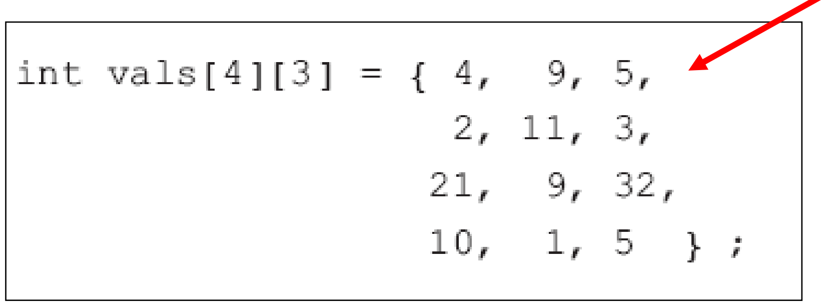
Initialize two-dimensional array

- Initialize an array by enclosing the initial values in **braces**

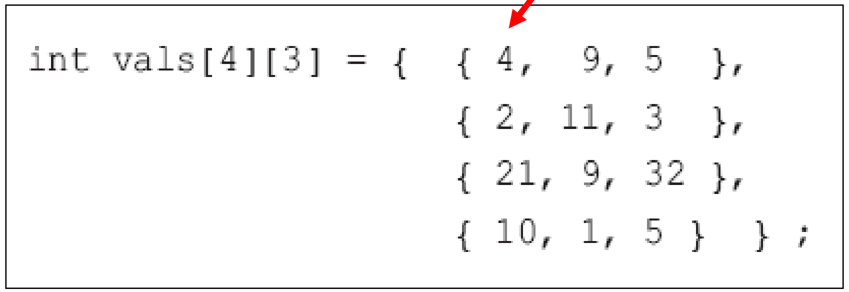


```
int vals[4][3] = { 4, 9, 5, 2, 11, 3, 21, 9, 32, 10, 1, 5 } ;
```

- Place the initial values of each row on a **separate line** or add **braces** in each row to improve readability



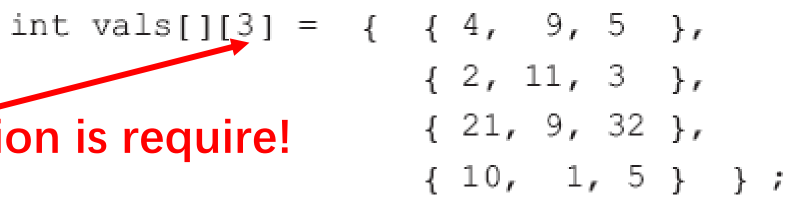
```
int vals[4][3] = { 4,  9, 5,
                   2, 11, 3,
                  21,  9, 32,
                  10,  1, 5  } ;
```



```
int vals[4][3] = {  { 4,  9, 5  },
                    { 2, 11, 3  },
                    { 21, 9, 32 },
                    { 10, 1, 5 }  } ;
```

Initialize two-dimensional array

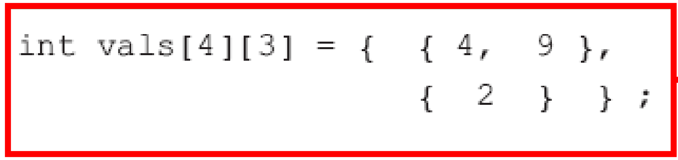
- Omit the first dimension
  - Compiler will calculate the number of rows



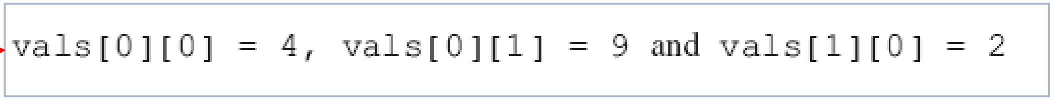
```
int vals[][3] =  {  { 4,   9, 5  },
                    { 2, 11, 3  },
                    { 21, 9, 32 },
                    { 10,   1, 5 }   } ;
```

Note that the first dimension is require!

- Missing values are initialized to 0



```
int vals[4][3] = {  { 4,   9 },
                    {  2   }   } ;
```

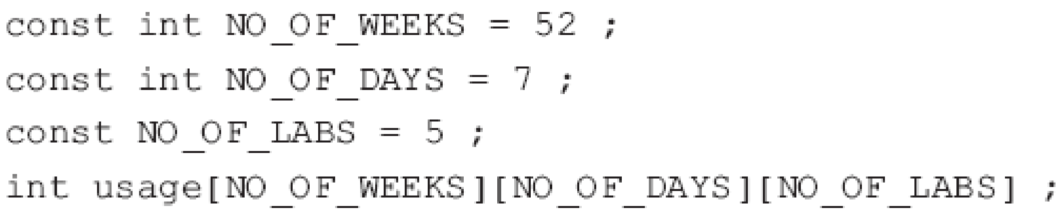


```
vals[0][0] = 4, vals[0][1] = 9 and vals[1][0] = 2
```

Row index	0	0	1
Column index	0	1	0

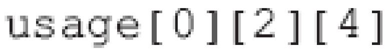
Multi-dimensional arrays

- Define arrays with any number of dimensions



```
const int NO_OF_WEEKS = 52 ;
const int NO_OF_DAYS = 7 ;
const NO_OF_LABS = 5 ;
int usage[NO_OF_WEEKS][NO_OF_DAYS][NO_OF_LABS] ;
```

- The elements of this array are accessed by using three subscripts



```
usage[0][2][4]
```

Structures

- Arrays are suitable for storing sets of **homogeneous data** (of the same type)
  - For example, a student’s test scores

Structures

- Arrays are suitable for storing sets of **homogeneous data** (of the same type)
  - For example, a student’s test scores

- For items of information that are **logically related** but each item may have a **different data type**?
  - For example, a student’s number (an integer) and five test scores (an array of scores)

Structures

- Arrays are suitable for storing sets of **homogeneous data** (of the same type)
  - For example, a student’s test scores

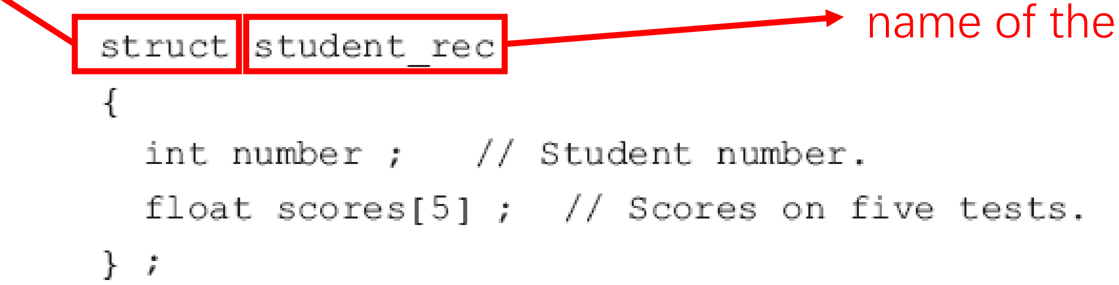
- For items of information that are **logically related** but each item may have a **different data type**?
  - For example, a student’s number (an integer) and five test scores (an array of scores)

- Logically related items of information that may have **different data types** can be combined into a **structure**

Declare a structure

- Step 1 : Declare a structure template:

A structure template consists of the reserved keyword **struct** followed by the **name of the structure**

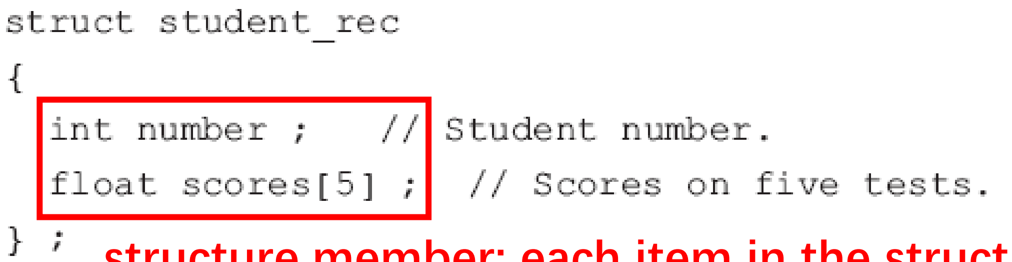


```
struct student_rec
{
  int number ;    // Student number.
  float scores[5] ;   // Scores on five tests.
} ;
```

name of the structure

Declare a structure

- Step 1 : Declare a structure template:

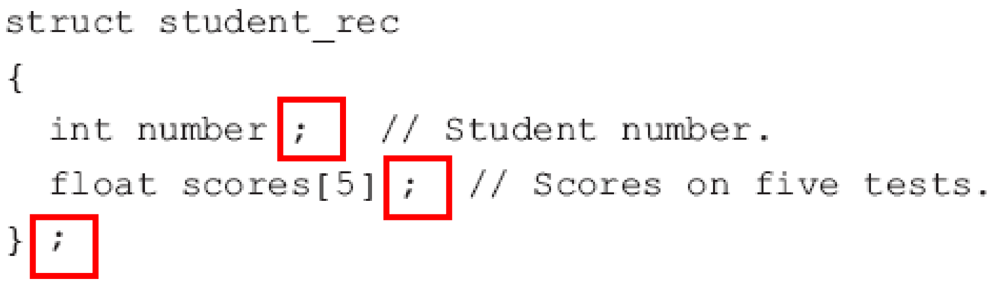


```
struct student_rec
{
  int number ;    // Student number.
  float scores[5] ;  // Scores on five tests.
} ;
```

structure member: each item in the structure

Declare a structure

- Step 1 : Declare a structure template:



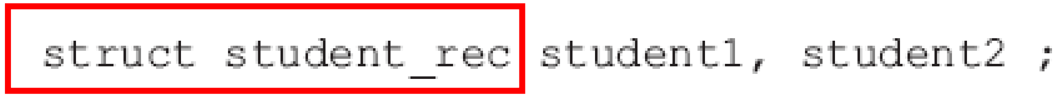
```
struct student_rec
{
  int number ;   // Student number.
  float scores[5] ; // Scores on five tests.
} ;
```

Don’t forget the semicolon!

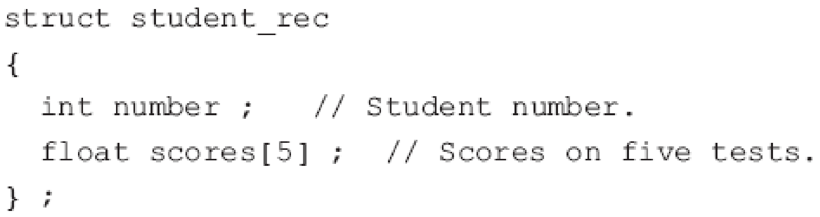
Declare a structure

- Step 2 : define variables with the type declared

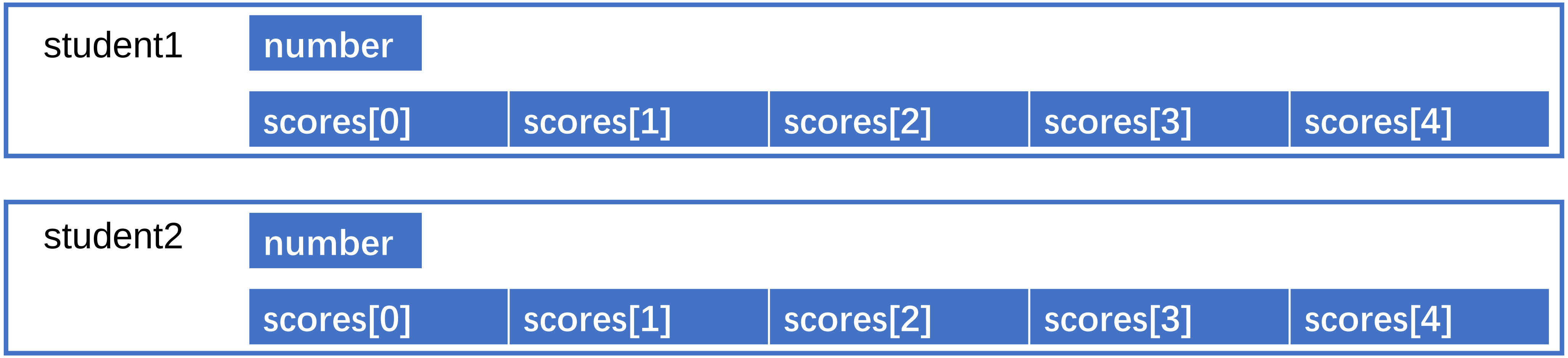
define student1 and student2 to be of the type **struct** student_rec



```
struct student_rec student1, student2 ;
```



```
struct student_rec
{
  int number ;    // Student number.
  float scores[5] ;  // Scores on five tests.
} ;
```



student1

number

scores[0] scores[1] scores[2] scores[3] scores[4]

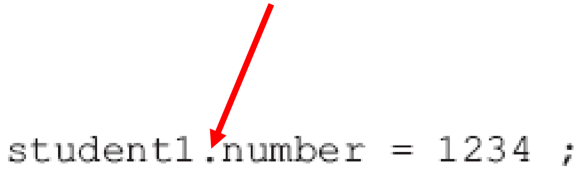
student2

number

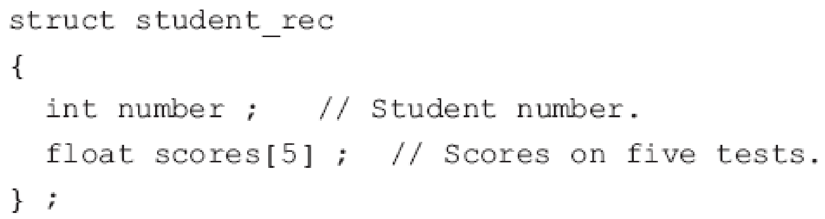
scores[0] scores[1] scores[2] scores[3] scores[4]

Declare a structure

- Can access members of a structure variable via the **member selection operator “.”**



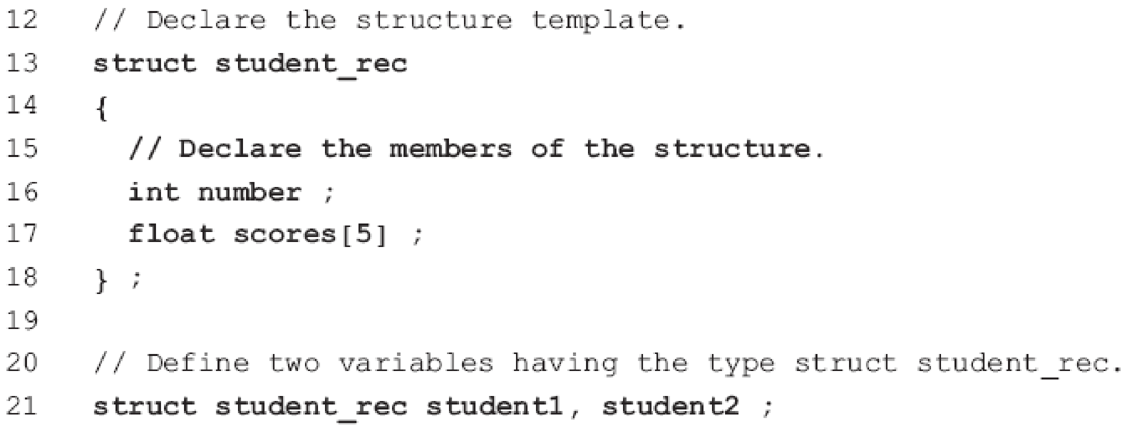
```
student1.number = 1234 ;
```



```
struct student_rec
{
  int number ;    // Student number.
  float scores[5] ;  // Scores on five tests.
} ;
```

Declare a structure

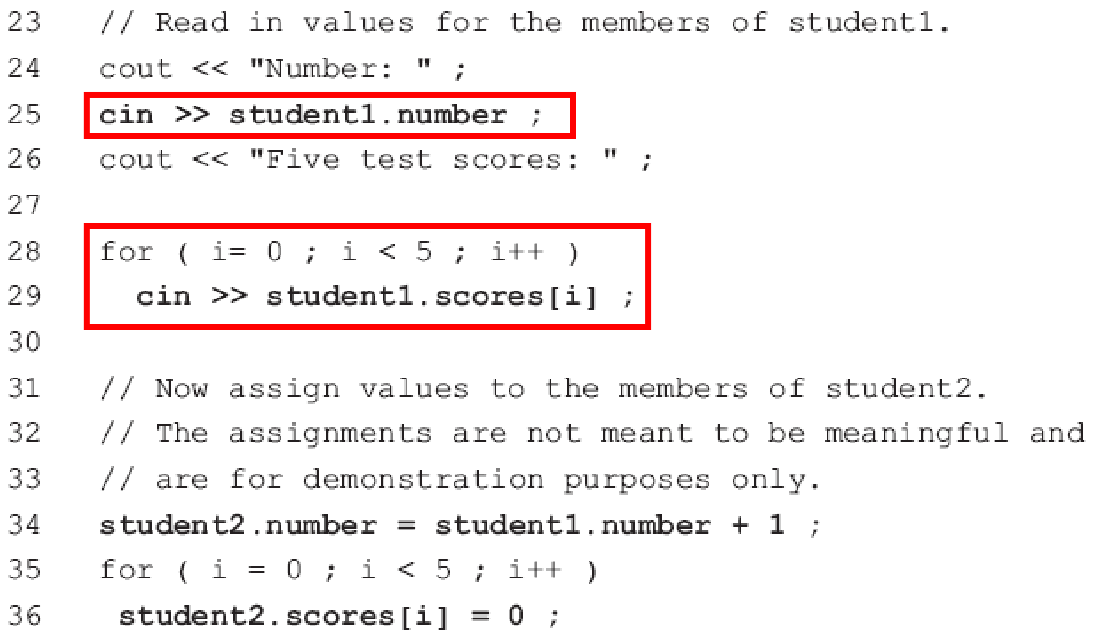
Example: inputs values for each member of a structure and displays it on the screen



```
// Declare the structure template.
struct student_rec
{
  // Declare the members of the structure.
  int number ;
  float scores[5] ;
} ;

// Define two variables having the type struct student_rec.
struct student_rec student1, student2 ;
```

Declare a structure

Example: inputs values for each member of a structure and displays it on the screen

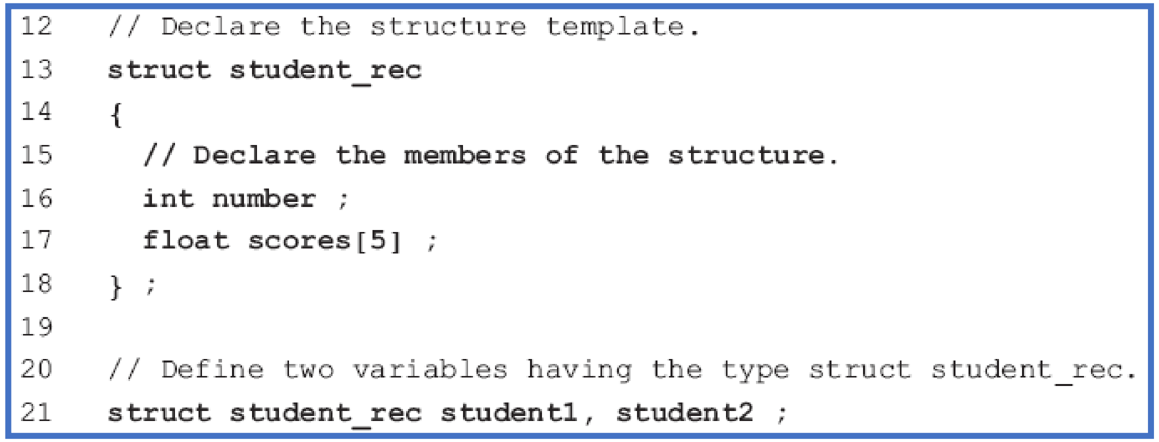


```
// Read in values for the members of student1.
cout << "Number: " ;
cin >> student1.number ;
cout << "Five test scores: " ;

for ( i= 0 ; i < 5 ; i++ )
  cin >> student1.scores[i] ;

// Now assign values to the members of student2.
// The assignments are not meant to be meaningful and
// are for demonstration purposes only.
student2.number = student1.number + 1 ;
for ( i = 0 ; i < 5 ; i++ )
 student2.scores[i] = 0 ;
```

Declare a structure

- Another declaration form of a structure:

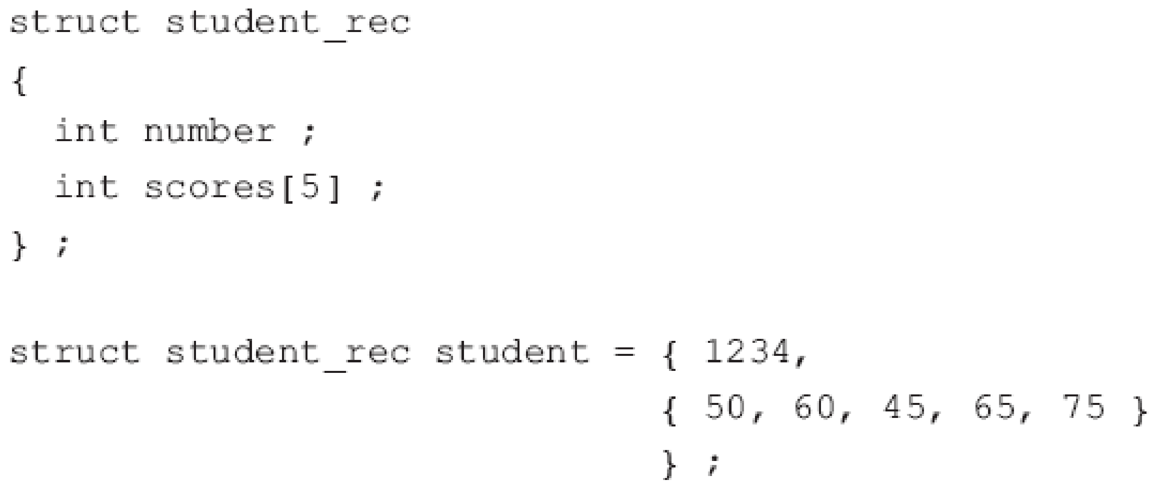
```
// Declaring a structure template without a structure tag.
struct    // No tag name after struct.
{
  int number ;
  float scores[5] ;
} student1, student2 ; // Variables follow immediately after the }.
```



```
// Declare the structure template.
struct student_rec
{
  // Declare the members of the structure.
  int number ;
  float scores[5] ;
} ;

// Define two variables having the type struct student_rec.
struct student_rec student1, student2 ;
```

Initialize a structure variable

- Place their initial values in **braces** to initialize a **structure**



```
struct student_rec
{
  int number ;
  int scores[5] ;
} ;

struct student_rec student = { 1234,
                             { 50, 60, 45, 65, 75 }
                             } ;
```

Nested structures

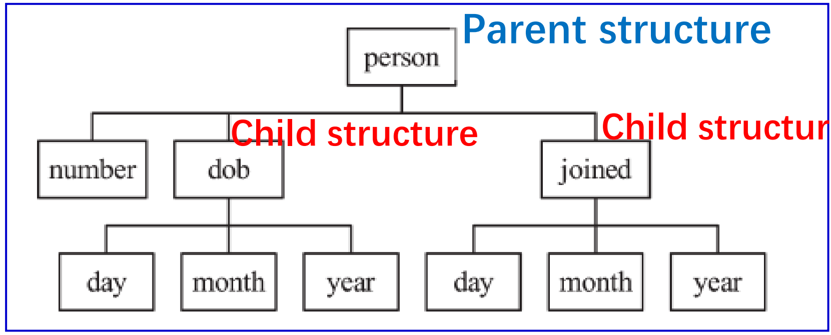
- A structure that contains another structure as one of its members.

Child structure

```
struct date
{
  int day ;
  int month ;
  int year ;
} ;
```

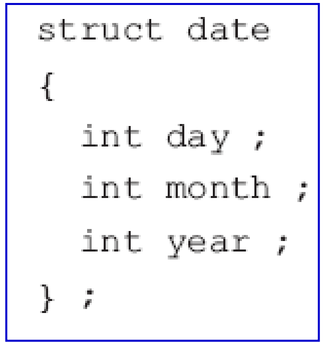
Parent structure

```
struct personnel  // Structure template for an employee.
{
  int number ;  // Employee number.
                // and various other structure members, e.g. pay.
  struct date dob ;     // The data type of dob is struct date.
  struct date joined ; // joined is also of type struct date.
} ;

struct personnel person ;
```

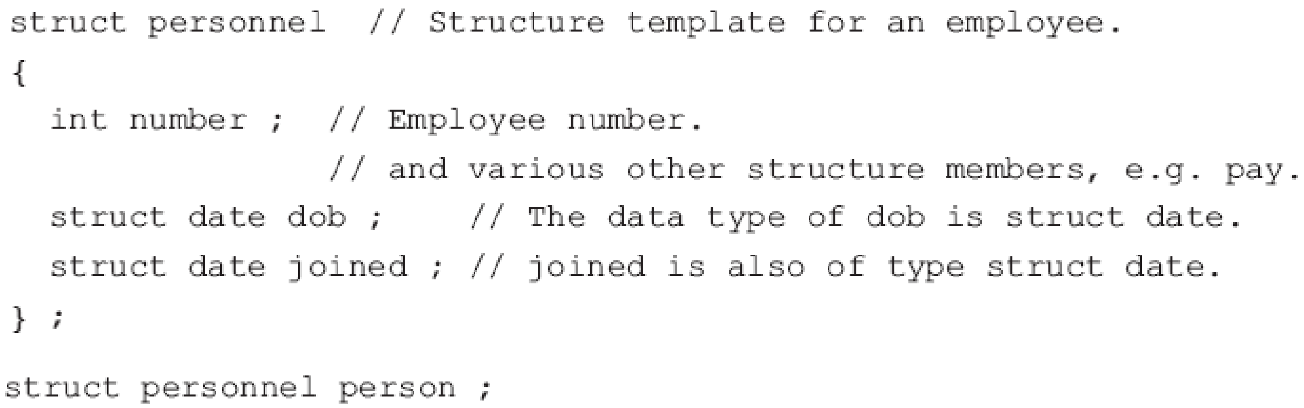


Nested structures

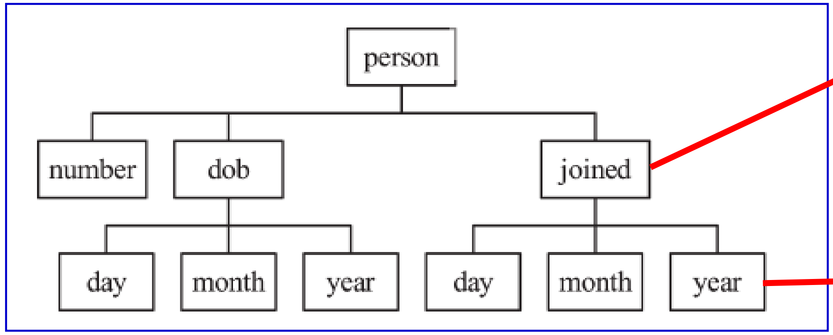
- A structure that contains another structure as one of its members.



```
struct date
{
  int day ;
  int month ;
  int year ;
} ;
```



```
struct personnel  // Structure template for an employee.
{
  int number ;  // Employee number.
                // and various other structure members, e.g. pay.
  struct date dob ;    // The data type of dob is struct date.
  struct date joined ; // joined is also of type struct date.
} ;

struct personnel person ;
```

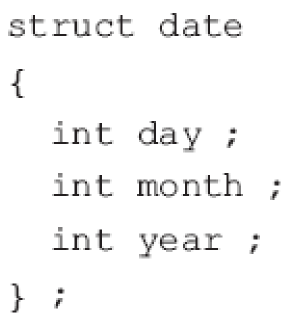


```
person.joined
```

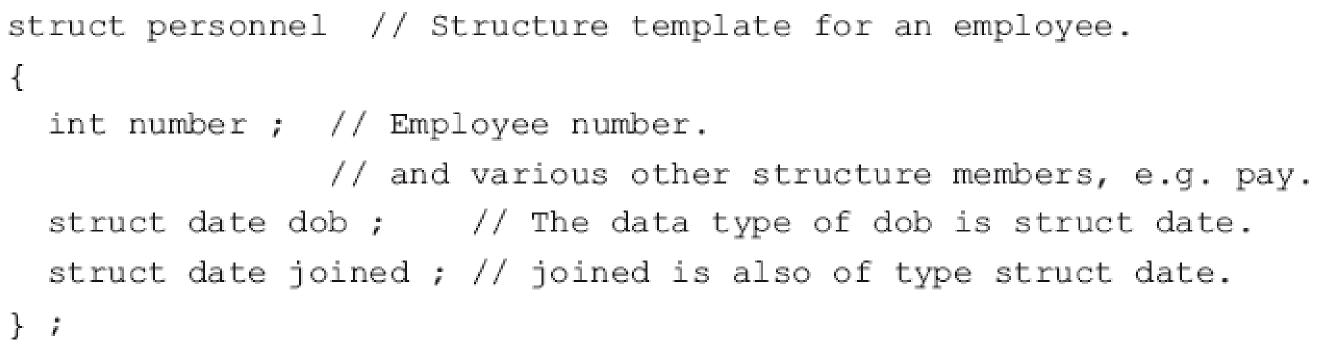
```
person.joined.year
```

The typedef statement

- **typedef** can define a **synonym for a built-in** or a **programmer defined** data type

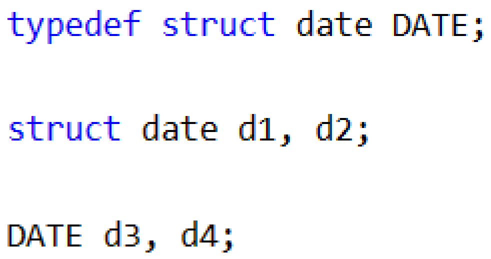


```
struct date
{
  int day ;
  int month ;
  int year ;
} ;
```



```
struct personnel  // Structure template for an employee.
{
  int number ;   // Employee number.
                 // and various other structure members, e.g. pay.
  struct date dob ;     // The data type of dob is struct date.
  struct date joined ; // joined is also of type struct date.
} ;
```

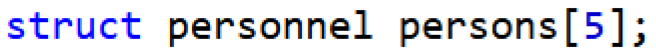
- Use **typedef** to define a synonym **DATE** for **struct** date:



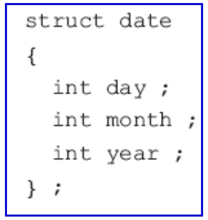
```
typedef struct date DATE;

struct date d1, d2;

DATE d3, d4;
```

Arrays of structures

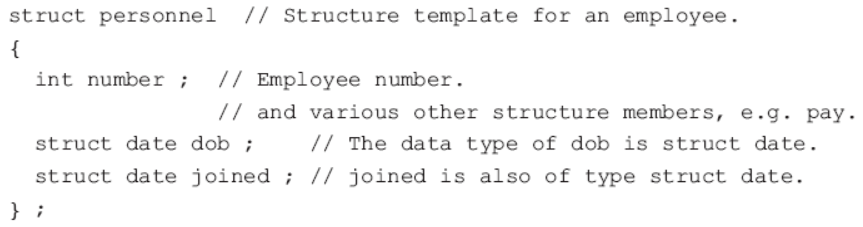
Example: Define a five-element array persons



```
struct personnel persons[5];
```



```
struct date
{
  int day ;
  int month ;
  int year ;
} ;
```



```
struct personnel  // Structure template for an employee.
{
  int number ;  // Employee number.
                // and various other structure members, e.g. pay.
  struct date dob ;    // The data type of dob is struct date.
  struct date joined ; // joined is also of type struct date.
} ;
```

Each element of this array is of the type struct personnel with members **number**, **dob** and **joined**

The members **dob** and **joined** are themselves structures and have members **day**, **month** and **year**

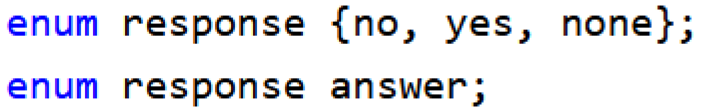
Which member will be accessed ?

persons[0].number ?

persons[4].joined.year ?

Enumerated data types

- An **enumerated** data type is used to **describe** a set of integer values



```
enum response {no, yes, none};
enum response answer;
```

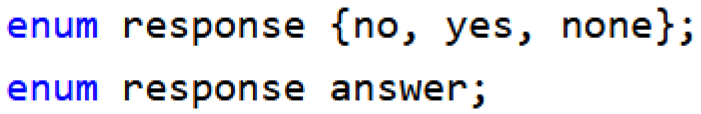
- These statements declare the data type response to have one of three possible values: no, yes, or none

- answer is defined as an enumerated variable of type response

Enumerated data types

- An **enumerated** data type is used to **describe** a set of integer values

Name of the enumerated data type defined as **response** is called the **enumeration tag**



```
enum response {no, yes, none};
enum response answer;
```

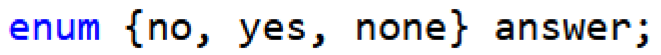
The names enclosed in **{** and **}** must be integer constants: The value of no is 0; the value of yes is 1; the value of none is 2

- These statements declare the data type response to have one of three possible values: no, yes, or none

- **answer** is defined as an enumerated variable of type **response**

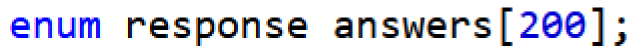
Enumerated data types

- Another definition form
  - When the enumerated data type and the enumerated variables are defined together, the enumeration tag is optional



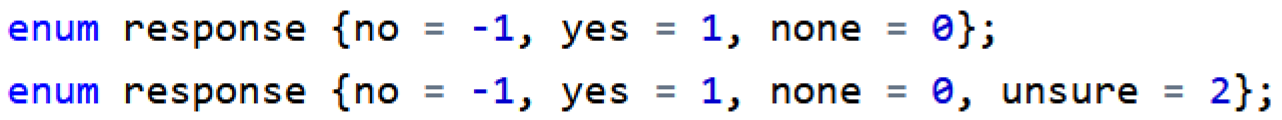
```
enum {no, yes, none} answer;
```

  - Arrays of enumerated data type



```
enum response answers[200];
```

  - Values other than 0, 1, and 2 can also be used

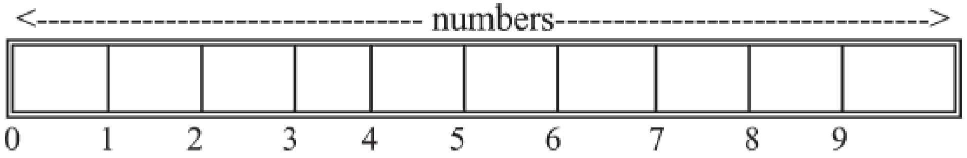


```
enum response {no = -1, yes = 1, none = 0};
enum response {no = -1, yes = 1, none = 0, unsure = 2};
```

Programming pitfalls

- The size (number of elements) of an array are placed between **brackets [ ]** and not between **parentheses ( )**

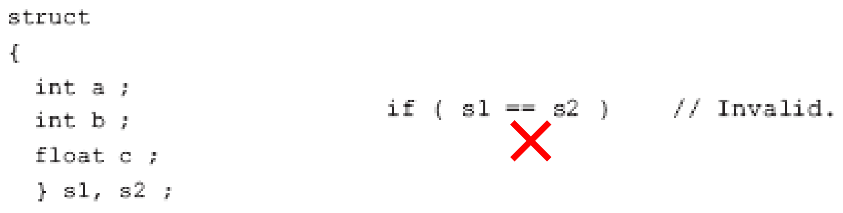
- The range of array index: from **0** to **N-1**, where N is its **number of elements**



<---------------------------------- numbers---------------------------------->
0
1
2
3
4
5
6
7
8
9

Programming pitfalls

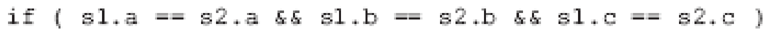
- You cannot compare structure variables in an **if statement**, even if they have the same structure template



```
struct
{
  int a ;
  int b ;
  float c ;
  } s1, s2 ;
```

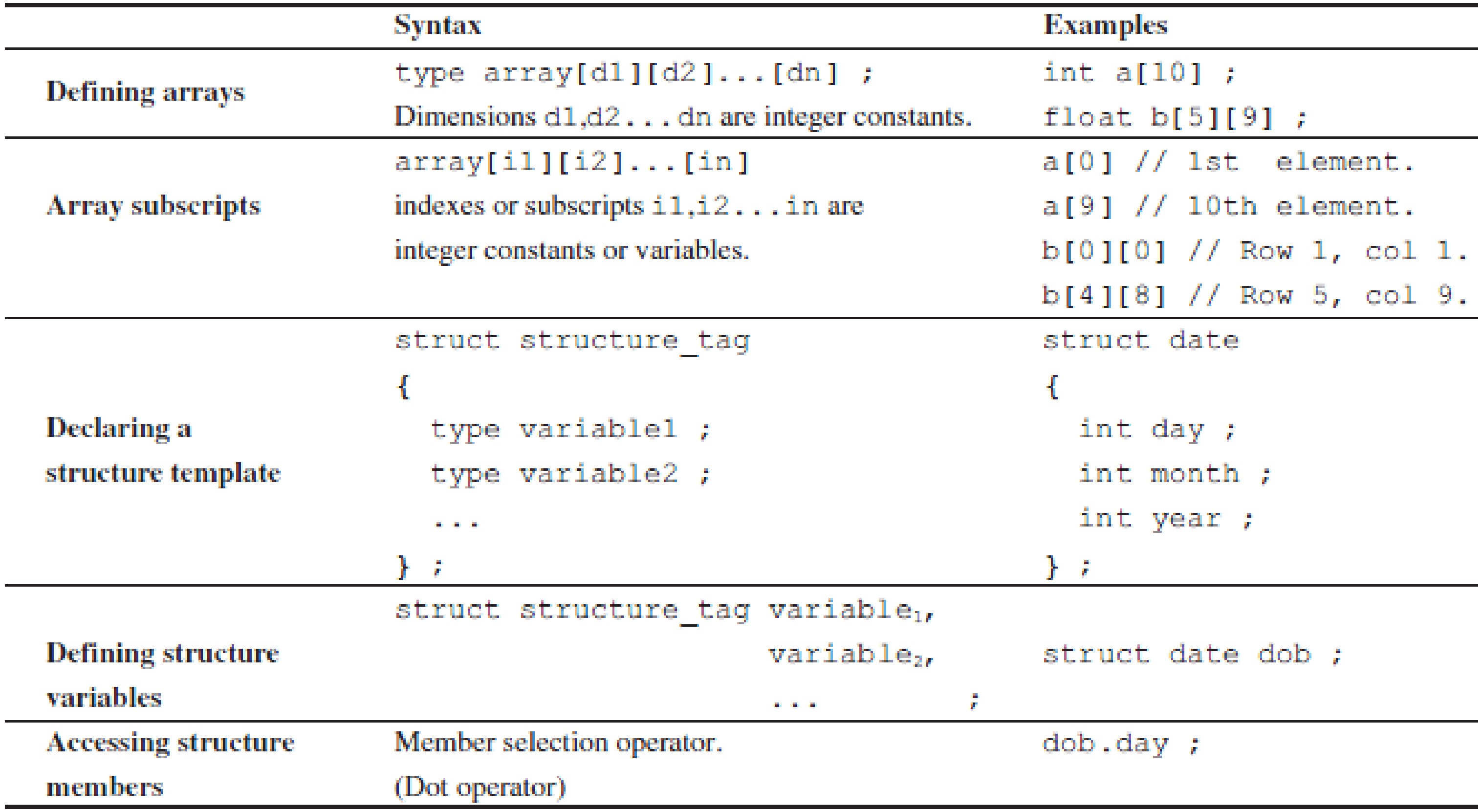
```
if ( s1 == s2 )     // Invalid.
```

- To test s1 and s2 for equality you must test each member of each structure for equality, as in the statement



```
if ( s1.a == s2.a && s1.b == s2.b && s1.c == s2.c )
```

Quick syntax reference



	Syntax	Examples

Defining arrays	`type array[d1][d2]...[dn] ;`
	Dimensions d1,d2. . . dn are integer constants.
		`int a[10] ;`
		`float b[5][9] ;`

Array subscripts	`array[i1][i2]...[in]`
	indexes or subscripts i1,i2. . . in are integer constants or variables.
		`a[0] // 1st  element.`
		`a[9] // 10th element.`
		`b[0][0] // Row 1, col 1.`
		`b[4][8] // Row 5, col 9.`

Declaring a structure template	`struct structure_tag`
	`{`
	`  type variable1 ;`
	`  type variable2 ;`
	`  ...`
	`} ;`
		`struct date`
		`{`
		`  int day ;`
		`  int month ;`
		`  int year ;`
		`} ;`

Defining structure variables	struct structure_tag variable$_1$, variable$_2$, ... ;	`struct date dob ;`
Accessing structure members	Member selection operator. (Dot operator)	`dob.day ;`

HOMEWORK

Homework 6

- 1. Write statements to define each of the following:

  (a) a one-dimensional array of floating-point numbers with ten elements

  (b) a one-dimensional array of characters with five elements

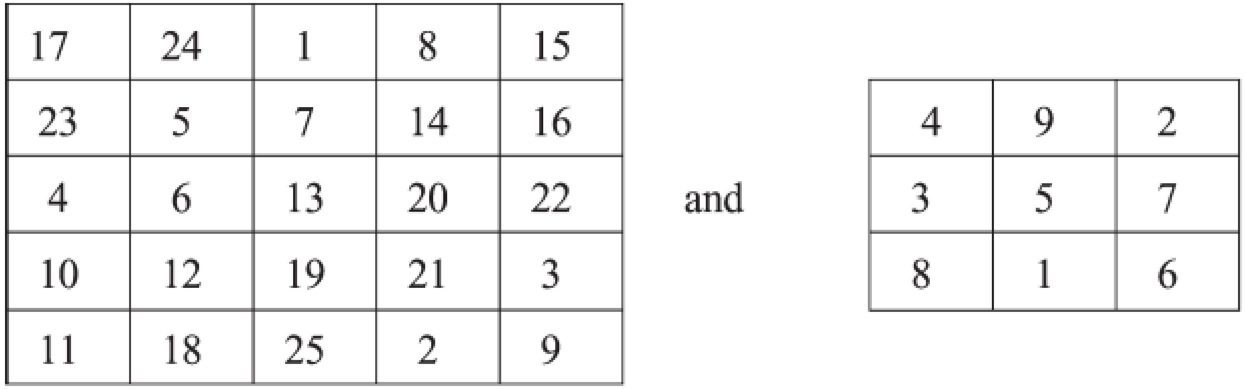
  (c) a two-dimensional array of integers with seven rows and eight columns

  (d) a 10 by 5 two-dimensional array of double precision numbers

  (e) a 10 by 8 by 15 three-dimensional array of integers.

Homework 6

- 2. In a magic square the **rows**, **columns**, and **diagonals** all have the same sum. For example:



17	24	1	8	15
23	5	7	14	16
4	6	13	20	22
10	12	19	21	3
11	18	25	2	9

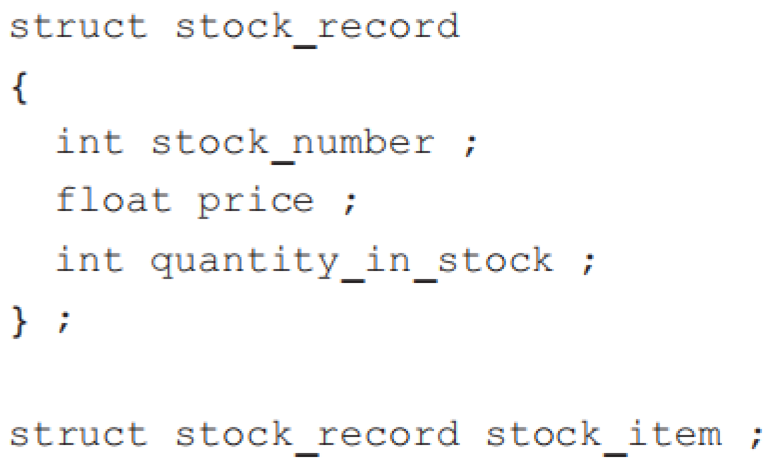
and

4	9	2
3	5	7
8	1	6

Write a program to read in a two-dimensional integer array and check if it is a magic square.

Homework 6

- 3. Given the following definitions,



```
struct stock_record
{
  int stock_number ;
  float price ;
  int quantity_in_stock ;
} ;

struct stock_record stock_item ;
```

write statements to

(a) assign a value to each member of stock_item

(b) input a value to each member of stock_item

(c) display the value of each member of stock_item

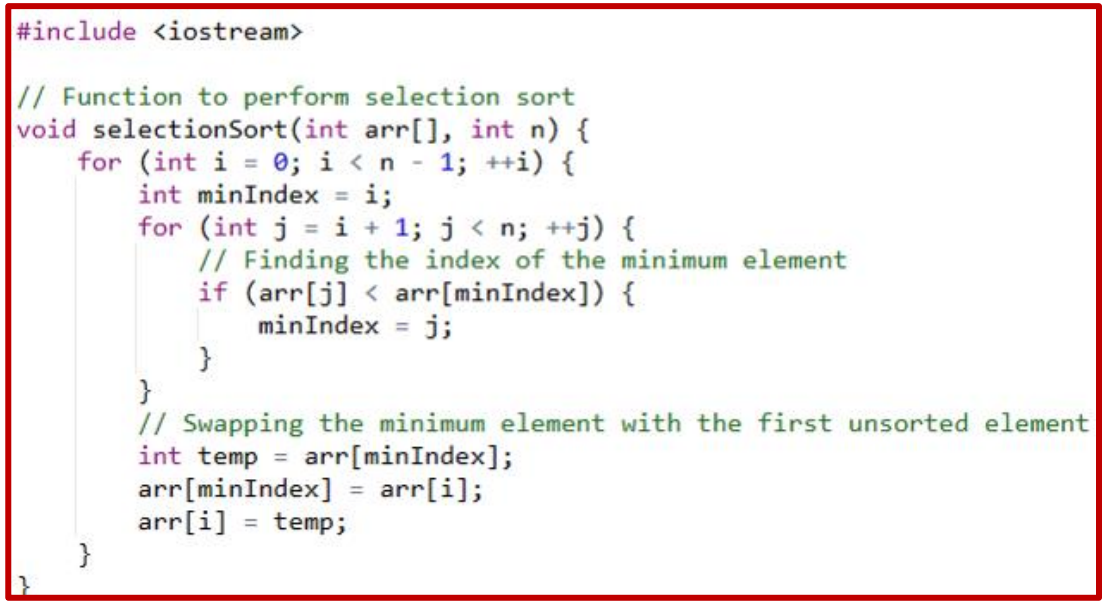
Homework 6

- 4. Create an enumerated data type for each of the following:
  - (a) the days of the week: Monday, Tuesday, Wednesday, and so on
  - (b) the months of the year
  - (e) the points on a compass (4 directions).

Homework 6

- 5. Given the array [22, 3, 1, 9, 6, 12, 8], print out the sorting results for **each round** of selection sort. For example, given the array [3, 2, 1], the sorting results for first round is [1, 2, 3].

Your can modify and utilize this function: ***selectionSort***



```cpp
#include <iostream>

// Function to perform selection sort
void selectionSort(int arr[], int n) {
    for (int i = 0; i < n - 1; ++i) {
        int minIndex = i;
        for (int j = i + 1; j < n; ++j) {
            // Finding the index of the minimum element
            if (arr[j] < arr[minIndex]) {
                minIndex = j;
            }
        }
        // Swapping the minimum element with the first unsorted element
        int temp = arr[minIndex];
        arr[minIndex] = arr[i];
        arr[i] = temp;
    }
}
```