



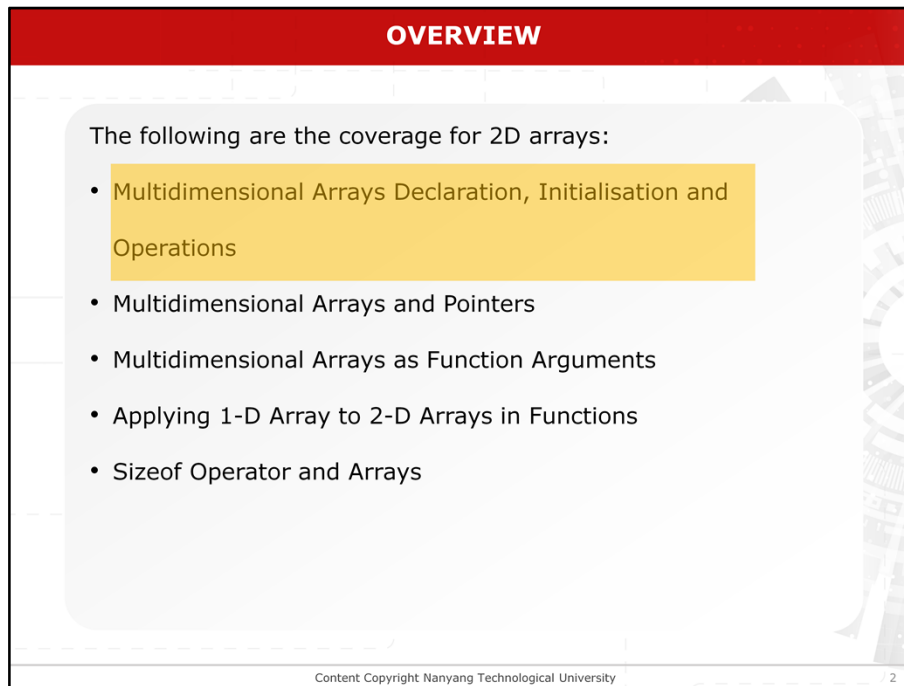
**NANYANG  
TECHNOLOGICAL  
UNIVERSITY**

## **CE1007/CZ1007 DATA STRUCTURES**

### **Lesson 7.1 Multidimensional Arrays Declaration, Initialisation and Operations**

Assoc Prof Hui Siu Cheung

**College of Engineering**  
School of Computer Science and Engineering



**OVERVIEW**

The following are the coverage for 2D arrays:

- Multidimensional Arrays Declaration, Initialisation and Operations
- Multidimensional Arrays and Pointers
- Multidimensional Arrays as Function Arguments
- Applying 1-D Array to 2-D Arrays in Functions
- Sizeof Operator and Arrays

Content Copyright Nanyang Technological University 2

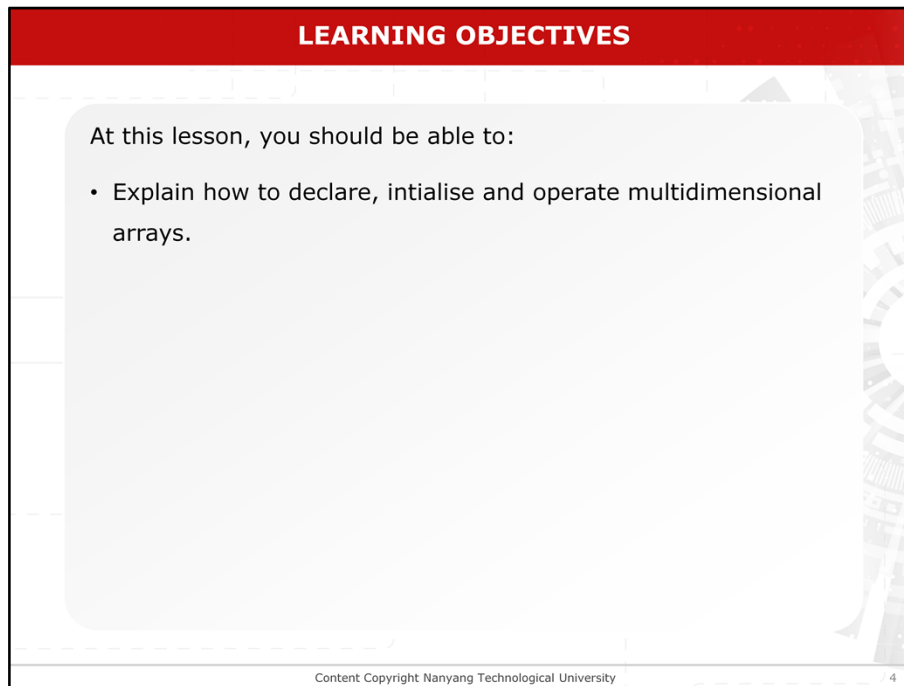
The following are the coverage for 2 dimensional ARRAYS. this video focusses on Multidimensional arrays declaration, initialization and operations.

**LEARNING OBJECTIVES**

At this lesson, you should be able to:

Content Copyright Nanyang Technological University 3

Learning Objectives: At this lesson, you should be able to:



**LEARNING OBJECTIVES**

At this lesson, you should be able to:

- Explain how to declare, initialise and operate multidimensional arrays.

Content Copyright Nanyang Technological University 4

Explain how to declare, initialise and operate multidimensional arrays.

The slide features a red header bar with the title "MULTIDIMENSIONAL ARRAYS DECLARATION" in white capital letters. Below the header, a light gray rounded rectangle contains a single bullet point: "Declared as consecutive pairs of brackets." The word "consecutive" is underlined and colored red. The background of the slide has a faint, stylized architectural pattern on the right side. At the bottom, a thin gray bar contains the text "Content Copyright Nanyang Technological University" on the left and the number "5" on the right.

Multi dimensional arrays can be declared as consecutive pairs of brackets.

### MULTIDIMENSIONAL ARRAYS DECLARATION

- Declared as **consecutive** pairs of brackets.
- E.g. **2-dimensional**; 3-element array of 5-element arrays:  

```
int x[3][5];
```

Content Copyright Nanyang Technological University 6

#### Multi-dimensional Arrays Declaration

A two-dimensional array can be declared as **3 element array of 5 element arrays**. Two indexes are needed to access each element of the array.

### MULTIDIMENSIONAL ARRAYS DECLARATION

- Declared as **consecutive** pairs of brackets.
- E.g. **2-dimensional**; 3-element array of 5-element arrays:  

```
int x[3][5];
```
- E.g. **3-dimensional**; 3-element array of 4-element arrays of 5-element arrays:  

```
char x[3][4][5];
```

Content Copyright Nanyang Technological University 7

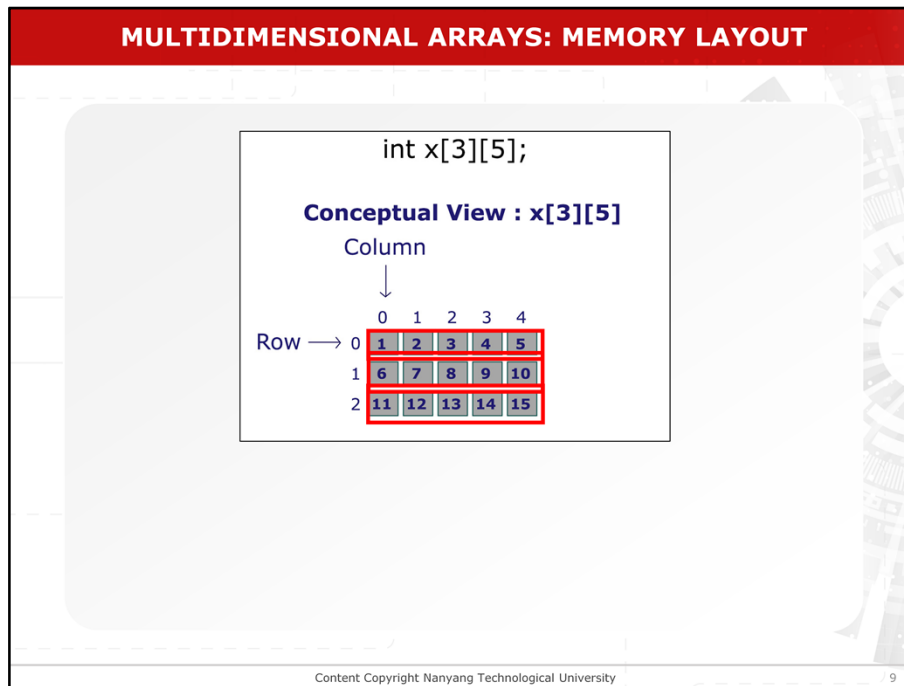
Similarly, a three-dimensional array can be declared as shown. Three indexes are used to access a specific element of the array.

MULTIDIMENSIONAL ARRAYS: MEMORY LAYOUT					
int x[3][5];					
	Column 0	Column 1	Column 2	Column 3	Column 4
Row 0	x[0][0]	x[0][1]	x[0][2]	x[0][3]	x[0][4]
Row 1	x[1][0]	x[1][1]	x[1][2]	x[1][3]	x[1][4]
Row 2	x[2][0]	x[2][1]	x[2][2]	x[2][3]	x[2][4]

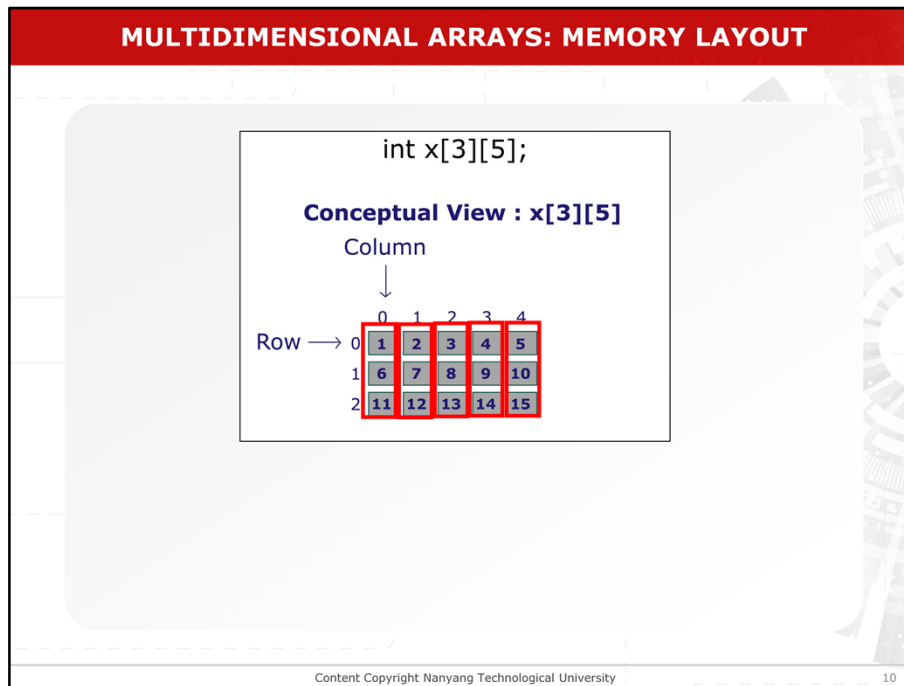
### Multi-dimensional Arrays: Memory Layout

The statement Declares a two-dimensional array of type **int** having three rows and five columns. The compiler will set aside the memory for storing the elements of the array. The two-dimensional array can also be viewed as a table made up of rows and columns.

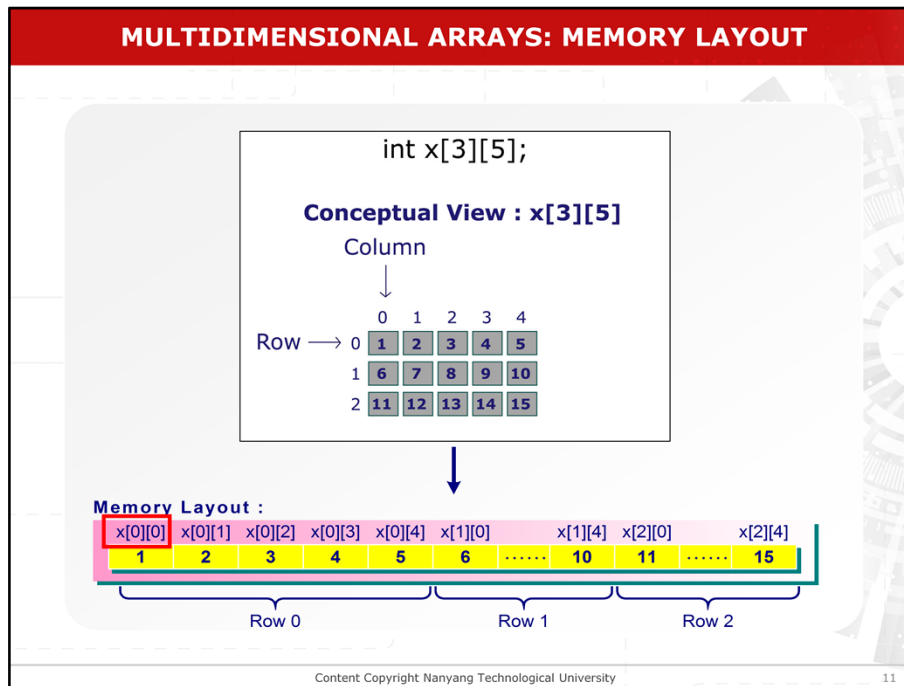




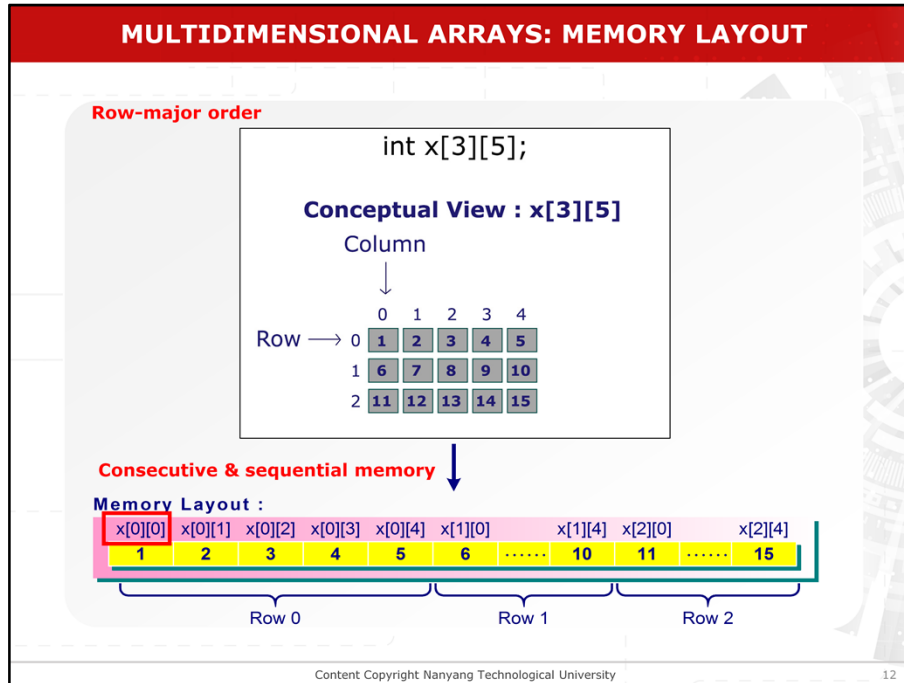
For example, the array can be represented as a table. The array consists of three rows



and five columns.



The array name and two indexes are used to represent each individual element of the array. The first index is used for the row, and the second index is used for column ordering. **x of zero zero** represents the first row and first column. and **x of 1 0** represents the second row and first column, and **x of 1 3** represents second row and fourth column, etc



A two-dimensional array is stored in *row-major* order in the memory. Note that the memory storage of the two-dimensional array **x** of **3 5** is consecutive and sequential.

**INITIALISING MULTIDIMENSIONAL ARRAYS**

- Initialising** multidimensional arrays: Enclose each row in braces.

```
int x[2][2] = { { 1, 2},    /* 1st row */  
              { 6, 7} }; /* 2nd row */
```

Content Copyright Nanyang Technological University 13

### Initializing Multi-dimensional Arrays

For the initialization of multi-dimensional arrays, each row of data is enclosed in braces:

The data in the first interior set of braces is assigned to the first row of the array, the data in the second interior set goes to the second row, etc. If the size of the list in the first row is less than the array size of the first row, then the remaining elements of the row are initialized to zero. If there are too many data, then it will be an error.

### INITIALISING MULTIDIMENSIONAL ARRAYS

- Initialising** multidimensional arrays: Enclose each row in braces.

```
int x[2][2] = { { 1, 2},      /* 1st row */  
               { 6, 7} }; /* 2nd row */  
or  
int x[2][2] = { 1, 2, 6, 7 };
```

Content Copyright Nanyang Technological University 14

Since the inner braces are optional, a multi-dimensional array can be initialized as shown.

### INITIALISING MULTIDIMENSIONAL ARRAYS

- Initialising** multidimensional arrays: Enclose each row in braces.

```
int x[2][2] = { { 1, 2},      /* 1st row */
               { 6, 7} }; /* 2nd row */
or
int x[2][2] = { 1, 2, 6, 7 };
```

**Partial** initialisation:

```
int exam[3][3] = {{1,2}, {4},{5,7}};
int exam[3][3] = { 1,2,4,5,7 };
int exam[3][3] = { {1,2,4}, {5,7}};
```

Content Copyright Nanyang Technological University

An array can also be initialized partially as shown. This statement initializes the first two elements in the first row, the first element in the second row, and the first two elements in the third row. All elements that are not initialized are set to zero by default.

### INITIALISING MULTIDIMENSIONAL ARRAYS

- Initialising** multidimensional arrays: Enclose each row in braces.

```
int x[2][2] = { { 1, 2},      /* 1st row */  
              { 6, 7} }; /* 2nd row */  
or  
int x[2][2] = { 1, 2, 6, 7 };
```

**Partial** initialisation:

```
int exam[3][3] = { {1,2}, {4},{5,7}};  
int exam[3][3] = { 1,2,4,5,7 };  
int exam[3][3] = { {1,2,4}, {5,7}};
```

Content Copyright Nanyang Technological University 16

The statement shown here is valid,



### INITIALISING MULTIDIMENSIONAL ARRAYS

- Initialising** multidimensional arrays: Enclose each row in braces.

```
int x[2][2] = { { 1, 2},      /* 1st row */
               { 6, 7} }; /* 2nd row */
or
int x[2][2] = { 1, 2, 6, 7 };
```

**Partial** initialisation:

```
int exam[3][3] = { {1,2}, {4},{5,7}};
int exam[3][3] = { 1,2,4,5,7 };
int exam[3][3] = { {1,2,4}, {5,7}};
```

Content Copyright Nanyang Technological University 17

but the two-dimensional array is initialized as shown here.

**INITIALISING MULTIDIMENSIONAL ARRAYS**

- Can **omit** the **outermost dimension** because compiler can figure that out. E.g.

```
int arr[ ][3][2] = { { {1,1},{0,0},{1,1}},
                    { {0,0},{1,2},{0,1}} };
```

gives a **[2]**[3][2] dimensioned array.

Content Copyright Nanyang Technological University

### Initializing Multi-dimensional Arrays

For **multi**dimensional arrays, we can omit the outermost dimension because the compiler can determine the dimension automatically. For example, the following array declaration creates a dimensioned array as shown

### INITIALISING MULTIDIMENSIONAL ARRAYS

- Can **omit** the **outermost dimension** because compiler can figure that out. E.g.

```
int arr[ ][3][2] = { { {1,1},{0,0},{1,1}},  
                    { {0,0},{1,2},{0,1}} };
```

gives a **[2][3][2]** dimensioned array.

The following is not correct. Why?

```
int wrong_arr[ ][ ] = {1,2,3,4};
```

Content Copyright Nanyang Technological University 19

However, the statement is incorrect, as the compiler is unable to determine the size of the array.

**OPERATIONS ON 2-D ARRAYS - USING ARRAY INDEX**

```
#include <stdio.h>
int main()
{ // declare an array with initialisation
  int array[3][3]={
    {5, 10, 15},
    {10, 20, 30},
    {20, 40, 60}
  };

  int row, column, sum;
  /* compute sum of row - traverse each row first */

  return 0;
}
```

**Nested Loop**

Content Copyright Nanyang Technological University

20

### Operations on 2-D Arrays

The program determines the sum of rows of two-dimensional arrays. It uses indexes to traverse each element of the 2 Dimensional **array**. In the program, the array is first initialized.

**OPERATIONS ON 2-D ARRAYS - USING ARRAY INDEX**

```

#include <stdio.h>
int main()
{
    // declare an array with initialisation
    int array[3][3]={
        {5, 10, 15},
        {10, 20, 30},
        {20, 40, 60}
    };

    int row, column, sum;

    /* compute sum of row - traverse each row first */
    for (row = 0; row < 3; row++)           // nested loop
    {
        /* for each row - compute the sum */
        sum = 0;
        for (column = 0; column < 3; column++)
            sum += array[row][column]; // using array indexes
        printf("Sum of row %d is %d\n", row+1, sum);
    }
    return 0;
}

```

**Nested Loop**

21

Content Copyright Nanyang Technological University

### Operations on 2-D Arrays

When accessing two-dimensional arrays using indexes, we use an index variable **row** to refer to the row number and another index variable **column** to refer to the column number.

### OPERATIONS ON 2-D ARRAYS - USING ARRAY INDEX

```

#include <stdio.h>
int main()
{ // declare an array with initialisation
  int array[3][3]={
    {5, 10, 15},
    {10, 20, 30},
    {20, 40, 60}
  };
  int row, column, sum;

  /* compute sum of row - traverse each row first */
  for (row = 0; row < 3; row++)           // nested loop
  {
    /* for each row - compute the sum */
    sum = 0;
    for (column = 0; column < 3; column++)
      sum += array[row][column]; // using array indexes
    printf("Sum of row %d is %d\n", row+1, sum);
  }
  return 0;
}

```

**Nested Loop**

Content Copyright Nanyang Technological University

22

A nested **for** loop is used to access the individual elements of the array. **Note that the first dimension of an array is row and the second dimension is column. It is row-major.**

## OPERATIONS ON 2-D ARRAYS - USING ARRAY INDEX

```

#include <stdio.h>
int main()
{ // declare an array with initialisation
  int array[3][3]={
    row    {5, 10, 15},
            {10, 20, 30},
            {20, 40, 60}
  };
  column

  int row, column, sum;

  /* compute sum of row - traverse each row first */
  for (row = 0; row < 3; row++) // nested loop
  {
    /* for each row - compute the sum */
    sum = 0;
    for (column = 0; column < 3; column++)
      sum += array[row][column]; // using array indexes
    printf("Sum of row %d is %d\n", row+1, sum);
  }
  return 0;
}

```

**Output**

Sum of row 1 is 30  
 Sum of row 2 is 60  
 Sum of row 3 is 120

**Nested Loop**

Content Copyright Nanyang Technological University
23

To process the sum of rows, we use the index variable **row** as the outer **for** loop. Then, **for each row, it traverses each element of each column and add them up to give the sum of rows.**

**OPERATIONS ON 2-D ARRAYS - USING ARRAY INDEX**

```

#include <stdio.h>
int main()
{ // declare an array with initialisation
  int array[3][3]={
    row    {5, 10, 15},
            {10, 20, 30},
            {20, 40, 60}
  };
  int row, column, sum;
  /* compute sum of each column */

  return 0;
}

```

Content Copyright Nanyang Technological University 24

### Operations on 2-D Arrays

The program determines the sum of columns of two-dimensional arrays. It uses indexes to traverse each element of the 2 Dimensional **array**. In the program, the array is first initialized.



**OPERATIONS ON 2-D ARRAYS - USING ARRAY INDEX**

```

#include <stdio.h>
int main()
{ // declare an array with initialisation
  int array[3][3]={
    row    {5, 10, 15},
            {10, 20, 30},
            {20, 40, 60}
  };

  int row, column, sum;

  /* compute sum of each column */
  for (column = 0; column < 3; column++)
  {
    sum = 0;
    for (row = 0; row < 3; row++)
      sum += array[row][column];
    printf("Sum of column %d is %d\n", column+1, sum);
  }
  return 0;
}

```

**Output**

Sum of column 1 is 35  
 Sum of column 2 is 70  
 Sum of column 3 is 105

Content Copyright Nanyang Technological University
25

### Operations on 2-D Arrays

To process the sum of columns, a nested **for** loop is used. We use the index variable **column** as the outer **for** loop. Then, **for each column, it traverses each element of each row and add them up to give the sum of columns**. Again note that the first dimension of an array is row and the second dimension is column. It is row-major.

**SUMMARY**

At this lesson, you should be able to:

- Explain how to declare, initialise and operate multidimensional arrays.

Content Copyright Nanyang Technological University 26

At this lesson, you should be able to explain how to declare, initialise and operate multidimensional arrays.