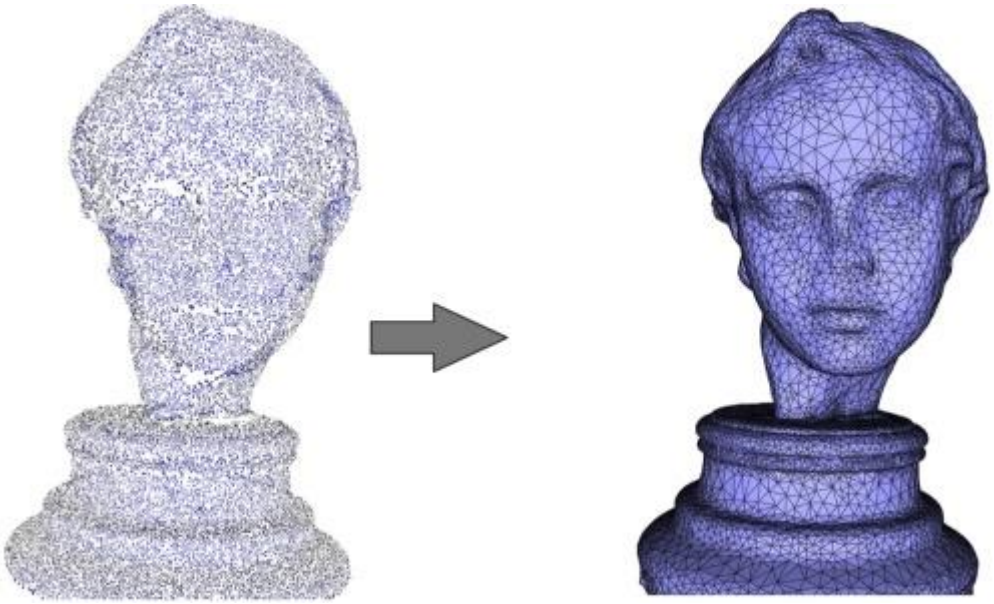
Computer programming (STQS1313)
Sem 2, Session 2020/2021
Notes - Arrays (Tatasusunan)

1) Topics:
   - One-dimensional array
   - Arrays as arguments
   - Two-dimensional arrays

2) We have been learning that each variable so far can be used to store only a single value at a time.

3) What will happen if we have a set of values, all having the same data type, for example:
   - daily amount spent by a student in a month (type double),
   - a list of marks for mathematics obtained by a class of students (type int/double),
   - a list of integer amounts,
   - a list of character codes,
   - a list of six floating-point prices.
   - point-cloud data in surface reconstruction → $P = \{(x_i, x_j, x_k) \vee 1 \leqslant i, j, k \leqslant N\}$



4) Arrays are used when you have a set of values, all having the same data type.

One-dimensional array

1) A simple list of containing items of the same data type is called a **one-dimensional array** (or a vector).

2) We will learn about how to **declare**, **initialise**, **store**, and **use** arrays.

3) Say that we have a list of amounts:

98 87 92 79 85

- declaration: `int a[5];`
- initialise:

```
a[0]=98; a[1]=87; a[2]=92; a[3]=79; a[4]=85;
```

- declaration and initialisation at once:

```
int amount[5]={98,87,92,79,85};
int amount[]={98,87,92,79,85};
```

- remember that the index/subscript starts from 0.

4) Array will be much more helpful, for example when you want to do element-wise algebraic operations on two arrays/vectors.

5) During array declaration each array is allocated enough memory to hold the number of data items specified in the declaration statement.

6) Each item in an array is called an **element** or **component** of the array. These elements are stored sequentially. An element's position is called as **index** or **subscript**. This means the 1st element is stored in the 1st reserved location an indexed as 0, and so on.

7) For example, say that you want to find the sum of elements stored in an array. So you can use

```
int a[5]={98,87,92,79,85};
sum = a[0] + a[1] + a[2] + a[3] + a[4];
```

or instead use a **for loop**:

```
int sum=0, N=5;
int a[N]={98,87,92,79,85};
for (int i=0; i<N; i++)
   sum = sum + a[i];
```

8) And, you can also "cout" the elements of the array using another loop

```
const int N=5;
int a[N]={98,87,92,79,85};
for (int i=0; i<N; i++)
   cout << a[i] << endl;
```

9) You can also use for loop with a "cin" for an array

```
const int N=5;
int a[N]={98,87,92,79,85};
for (int i=0; i<N; i++)
   cin >> a[i];
```

10) Example 7.1: A program on the input and output techniques using an array and a for loop.

11) Example 7.2: Finding the sum of all elements in an array

12) Array of various types:

- `int amount[5]={98,87,92,79,85};`
- `char code[]={'s','a','m','p','l','e'};`
- `string item[]={"bulbs","papers","hammers"};`

which also can be initialized in its declaration statement.

13) Example: Finding the maximum value between elements in an array

```
#include<iostream>
using namespace std;
int main()
{
    int maxnum, a[5] = {2, 18, 1, 27, 16};
    maxnum = a[0];
    for(int i=1; i<5; i++)
        if (maxnum<a[i])
            maxnum = a[i];

    cout << "The maximum value is " << maxnum << endl;
    return 0;
}
```

whose process can be described by the table below:

Iteration	`i`	Value of `a[i]`	Expression in `if (maxnum< a[i])`	Value in `maxnum`
0	-	-	-	2
1	1	18	`maxnum<a[i]` is true	18
2	2	1	`maxnum<a[i]` is false	18
3	3	27	`maxnum<a[i]` is true	27
4	4	16	`maxnum<a[i]` is false	27

14) Example: Finding the minimum value between elements in an array

```
#include<iostream>
using namespace std;
int main()
{
    int minnum, a[5] = {2, 18, 1, 27, 16};
    minnum = a[0];
    for(int i=1; i<5; i++)
        if (minnum<a[i])
            minnum = a[i];

    cout << "The minimum value is " << minnum << endl;
    return 0;
}
```

whose process can be described by the table below:

Iteration	`i`	Value of `a[i]`	Expression in `if (minnum>a[i])`	Value in `minnum`
0	-	-	-	2
1	1	18	`minnum>a[i]` is false	2
2	2	1	`minnum>a[i]` is true	1
3	3	27	`minnum>a[i]` is false	1
4	4	16	`minnum>a[i]` is false	1

Array as function arguments

1) **Array elements** are passed to a called function in the same manner as scalar/regular variables.

2) For example, if we call `findMax(grades[2],grades[6])`, this means that the calling function will pass the value of `grades[2]` and `grades[6]` to the function `findMax`.

3) We can also call the **whole array**, for example `findSum(grades)` to find the sum of all elements in an array named `grades`. In this case, we need to specify an array in the function declaration.

4) Example: using **array elements** as function arguments (predefined function) - finding the cosine value of array `x`, and stored as a new array `y`.

```
#include<iostream>
#include<cmath>
#include<iomanip>
using namespace std;
int main()
{
      const double PI = 22.0/7;
      double x[5] = {0, PI/2, PI, 3*PI/2, 2*PI}, y[5];
      for(int i=0; i<5; i++)
            y[i] = cos(x[i]);

      cout << fixed << showpoint << setprecision(2);
      for(int i=0; i<5; i++)
      cout << "For i=" << i << ", x[i]=" << x[i];
           << " and cos(x[i])=" << y[i] << endl;

      return 0;
}
```

5) Example: using **array elements** as function arguments (user-defined function) – element-wise summation of two arrays

```
#include<iostream>
using namespace std;
int findSum(int x, int y);
int main()
{
      int a[5] = {2, 18, 1, 27, 16};
      int b[5] = {1, 2, 3, 4, 5};
      int c[5];

      for(int i=0; i<5; i++)
            c[i] = findSum(a[i],b[i]);

      cout << "The sum of vectors a and b is" << endl;
      for(int i=0; i<5; i++)
            cout << c[i] << " ";
      return 0;
}

int findSum(int x, int y)
{
      return (x+y);
}
```

6) Example: passing a **complete (whole) array** to a function – finding the maximum value in an array (compare with Code 39):

```
#include<iostream>
using namespace std;
int findMax(int [5]);
int main()
{
    int maxnum, a[5] = {2,18, 1, 27, 16};

    maxnum = findMax(a);
    cout << "The maximum value is " << maxnum << endl;

    return 0;
}

int findMax(int b[5])
{
    int m = b[0];
    for(int i=1; i<5; i++)
        if (m < b[i])
            m = b[i];
    return m;
}
```

7) Note that when **calling the function** `findMax`, we don't need to write the size of the function `N`, hence `findMax(a)` is sufficient.
8) It is also **not necessary to state the array size `N` in function declaration and also in function header**. This is because, array `a` is created in `main()` and no additional storage space is needed in `findMax()`, hence declaration for `findMax()` and its corresponding function header can omit the array size. Therefore, the following code is valid.

```
#include<iostream>
using namespace std;
int findMax(int b[]);
int main()
{
    int maxnum, a[5] = {2,18, 1, 27, 16};

    maxnum = findMax(a);
    cout << "The maximum value is " << maxnum << endl;

    return 0;
}

int findMax(int b[])
{
    int m = b[0];
    for(int i=1; i<5; i++)
        if (m< b[i])
            m = b[i];

    return m;
}
```

9) In fact, in **function declaration**, we are allowed to write `int findMax(int [])` which avoids stating both the name of array and its size.
10) However, in many cases, the size of an array `N` is usually needed in the `main` function or other functions. This is shown below.

```
#include<iostream>
using namespace std;
const int N=5;
int findMax(int []);          // can also use b[N], b[], []
int main()
{
    int maxnum, a[N] = {2,18, 1, 27, 16};  // can also use a[]=...

    maxnum = findMax(a);
    cout << "The maximum value is " << maxnum << endl;

    return 0;
}

int findMax(int b[])
{
    int m = b[0];
    for(int i=1; i<N; i++)
        if (m< b[i])
            m = b[i];

    return m;
}
```

11) Note that from the above code, we define the arrays size `N` as a global constant, which means that `N` can be used in any functions including the `main` function and `findMax` function.

12) If the array size N is declared locally (not globally), then we need to pass the array size N to the called function in the function call. This is shown below:

```
#include<iostream>
using namespace std;
int findMax(int [], int);
int main()
{
      const int N=5;
      int maxnum, a[N] = {2,18, 1, 27, 16};
      maxnum = findMax(a,N);
      cout << "The maximum value is " << maxnum << endl;

      return 0;
}

int findMax(int b[], int n)
{
      int m = b[0];
      for(int i=0; i<n; i++)
            if (m < b[i])
                  m = b[i];
      return m;
}
```

7) Recall that when a **single scalar or an array element** is used as an argument of a function, the called function receives only a copy of the passed value.
8) But, when the **whole array** is used as an argument of a function, the called function is given direct access to the original array. In this way, any changes the called function makes are made directly to the array.
9) This is shown in the following code.

```
#include<iostream>
using namespace std;
const int N=5;
void change(int []);          // can also use b[N], b[], []
int main()
{
    int a[] = {2,18, 1, 27, 16};  // can also use a[]=...

    cout << "Before using the function change(), a is: ";
    for(int i=0; i<N; i++)
        cout << a[i] << " ";

    change(a);

    cout << "\n\nAfter using the function change(), a is: ";
    for(int i=0; i<N; i++)
        cout << a[i] << " ";
    cout << endl;
    return 0;
}

void change(int b[])
{
    b[0] = 1;
    b[3] = 2;
}
```

- Note that in the above code, the first and fourth elements of array `a` are changed by the function named `change`.
- This has affected two elements, implying that we can't use value-returning functions such as those of type `int`, `double` and so on. Hence, `change` must be of type `void`.
- Since passing the whole array gives the called function a direct access to the original array `a`, we **don't need to add ampersand & (as used in pass-by-reference) when declaring b** in function declaration and function header.
- This is shown in the following code, which contains slight changes of the element-wise summation of two arrays from 5.

```
#include<iostream>
using namespace std;
const int N=5;
void findSum(int [], int [], int []);
int main()
{
	int a[5] = {2, 18, 1, 27, 16};
	int b[5] = {1, 2, 3, 4, 5};
	int c[5];

	findSum(a,b,c);

	cout << "The sum of vectors a and b is" << endl;
	for(int i=0; i<5; i++)
		cout << c[i] << " ";
	return 0;
}

void findSum(int x[], int y[], int z[])
{
	for(int i=0; i<N; i++)
		z[i] = x[i] + y[i];

}
```

- Note that array `c` (along with arrays `a` and `b`) is included in the function call even though `c` is supposed to store the output of the summation of `a` and `b`.

Two-dimensional arrays

1) Sometimes referred to as a *table*.

2) A two-dimensional array: consists of both rows and columns of elements.

3) A 3-by-4 array of type interger named `val` can be **declared** by

```
int val[3][4];
```

where the number in the first square bracket is the number of rows, and the number in the second square bracket is the number of columns.

4) Unlike the one-dimensional array, the number of rows and columns must be specified during array declaration.

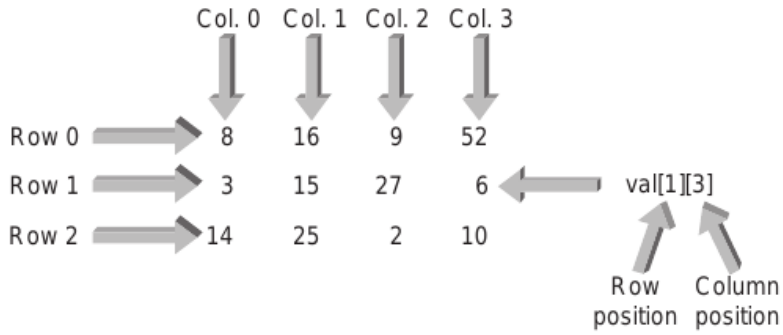
5) Declaration and initialization at the same time:

```
int val[3][4] = {{8,16,9,5},{3,15,27,6},{14,25,2,10}};
```

or

```
int val[3][4] = {8,16,9,5,3,15,27,6,14,25,2,10};
```

6) When using the array `val` above, the term `val[1][3]` identifies the element of array `val` in row 1, column 3 (note that row and column start from index 0), which is 6. This is shown in the figure below:



7) Example: displaying a two-dimensional array by (i) element notation, and (ii) by using nested for loops. See Code 47.
   - nested loops are especially useful when dealing with two-dimensional arrays because they allow the programmer to designate and cycle through each element easily.
   - In the program, the variable `i` controls the outer loop which corresponds to the rows, and the variable `j` controls the inner loop which corresponds to the columns.

8) After two-dimensional array elements have been assigned, array processing can begin. For example, we can multiply a two-dimensional array named `val`, say, with 10. This is shown in Code 48.

9) We can also add two arrays of the same size. This is shown in Code 49.

10) Using the **elements of two-dimensional array as function argument** is possible, and it is identical to passing elements of one-dimensional array. The program in Code 50 is similar to Code 48 where we try to multiply the elements of the array with 10, but by using function whose input is the elements of an array:

```
#include<iostream>
#include<iomanip>
using namespace std;
int multFun(int );
int main()
{
    const int NROW = 3;
    const int NCOL = 4;

    int i, j;
    int val[NROW][NCOL] = {8,16,9,5,3,15,27,6,14,25,2,10};

    for (i=0; i<NROW; i++)
    {
        cout << endl;             // print a new line for each row
        for (j=0; j<NCOL; j++)
        {
            val[i][j] = multFun(val[i][j]);
            cout << setw(4) << val[i][j];
        }
    }
    cout << endl;

    return 0;
}

int multFun(int b)
{
    b = b*10;
}
```

11) Using the **whole two-dimensional array as function argument** is also possible, and in fact it is identical to passing one-dimensional array. The program in Code 51 is similar to Code 48 where we try to multiply the elements of the array with 10, but by using function whose input is the whole array:

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

const int NROW = 3;
const int NCOL = 4;
void multFun(int [NROW][NCOL]);

int main()
{
      int val[NROW][NCOL] = {{8,16,9,5},{3,15,27,6},{14,25,2,10}};
      multFun(val);

      return 0;
}

void multFun(int a[NROW][NCOL])
{
      for (int i=0; i<NROW; i++)
      {
            cout << endl;          // print a new line for each row
            for (int j=0; j<NCOL; j++)
      {
            a[i][j] = a[i][j]*10;
                  cout << setw(4) << a[i][j];
            }
      }
      cout << endl;
}
```

12) The program from Code 52 below is similar to Code 48. It tries to add two arrays and display the output, but using function where the whole two arrays are arguments for the function.

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

const int NROW = 2;
const int NCOL = 3;
void addFun(int [NROW][NCOL], int [NROW][NCOL]);

int main()
{
int a[NROW][NCOL] = {{1,3,5},{7,9,11}};
int b[NROW][NCOL] = {{0,2,4},{6,8,10}};

addFun(a,b);

return 0;
}

void addFun(int x[NROW][NCOL], int y[NROW][NCOL])
{
      int z[NROW][NCOL];
      for (int i=0; i<NROW; i++)
      {
            cout << endl;           // print a new line for each row
            for (int j=0; j<NCOL; j++)
            {
                  z[i][j] = x[i][j] + y[i][j];
                  cout << setw(4) << z[i][j];
            }
      }
cout << endl;
}
```

13) Larger dimensional array:

- aren't commonly used in C++
- but it is still allowed by C++
- Example, for declaration: `int response[4][10][6];`
- think of a three-dimensional array as a book of data tables:
  i. the first subscript value as the *row*,
  ii. the second subscript value as the *column*,
  iii. the third subscript value as the *page number* of the selected table of rows and columns (often called as "rank").
- similarly, array of any dimension can be declared and used in C++.