

Some More Arithmetic Operators

- We have already met the four basic arithmetic operators + - * /
- there are some other useful operators

+= -= *= /=

- these are used as a shorthand in the following way

```
x += 5;  means  x = x + 5;
x -= 5;  means  x = x - 5;
x *= 5;  means  x = x * 5;
x /= 5;  means  x = x / 5;
```

x could be either float or integer

- there are two other operators which can only be applied to integer variables

the increment operator ++
the decrement operator --

- they are used to increment or decrement the value of a variable by 1
- they can be placed before or after the integer variable

x++; and ++x;

both have the same result - x is incremented by 1

- you can include ++ and -- in more complicated statements but then it does make a difference whether they are placed before or after the variable

k = x++; sets k equal to x and then increments x

whereas

k = ++x; increments x and then sets k equal to x

for loops - how to repeat a set of instructions a specific number of times

```
#include <iostream.h>

int main()
{
    int i;

    for(i=0; i<100; i++)
    {
        cout << "the value of i is " << i << endl;
    }

    return 0;
}
```

- this program repeats the command between the curly brackets 100 times
- the `for` loop uses an integer counter, in this case called `i`
- the `for` statement has three parts separated by semicolons

the first part is executed at the first iteration: it initialises the counter

the second part is a logical test carried out at the start of every iteration: if the test is false the loop terminates otherwise it continues

the third part is carried out at the end of each iteration: it updates the value of the counter

- in the above loop the counter `i` is initialised to zero and then incremented by 1 at each iteration until it reaches the value 100 when the loop stops

Some more `for` loops

- other versions of the `for` loop are possible

- `for(i=-10; i<100; i++)`

would initialise the counter to -10. This loop would have 110 iterations

- `for(i=0; i<100; i += 2)`

would increment the counter by 2 at each iteration. This loop would have 50 iterations

- `for(i=100; i>0; i--)`

would initialise the counter to 100 and then decrement it by 1 at each iteration until it reached 0.

- many other forms are possible

An Example using a for loop

- this program adds together the integers 1 to 100

```
#include <iostream.h>

int main()
{
int i, sum=0;

for(i=1; i<101; i++)
    sum += i;

cout << "the sum is " << sum << endl;

return 0;
}
```

- this program uses a variable `sum` to accumulate the required value step by step
- `sum` is initialised to zero.
- the counter `i` is used to represent each of the integers 1 to 100 in turn
- `i` is initialised to 1
- at each iteration `i` is added to `sum` then `i` is incremented by 1
- after 100 iterations `sum` will now be equal to the required value

Arrays

- arrays are collections of variables all of the same type and stored next to one another in the computer's memory
- arrays are useful for example for storing statistical data e.g. the salaries of employees in a company, the rainfall on each day of a year or populations of different towns in a country.
- arrays are defined in a way similar to variables.

```
int arr[10];
```

defines an array called `arr` which has 10 members. Each member is an integer variable.

- in the definition statement the size of the array is given in square brackets `[]`
- after it has been defined you can treat each member of the array like a variable. You can refer to each individual member by using an index which describes its position in the array.
- the first member of an array has index 0. So for example the ten members of the above array `arr` would be

```
arr[0] arr[1] arr[2] arr[3] arr[4]  
arr[5] arr[6] arr[7] arr[8] arr[9]
```

- in general if an array has N members then the first member has index 0 and the last has index $N-1$
- note that the index number is also given in square brackets `[]`
- you can assign values to each member just as with other variables

```
arr[0] = 10;  
arr[3] = 3;
```

would set the first member of `arr` equal to 10 and the fourth equal to 3.

Initialising an Array

- You can initialise an array at the same time as you are defining it as follows

```
int arr[5] = {11,2,6,7,3};
```

- Alternatively you could input data from the keyboard

```
for(i=0; i<5; i++)  
{  
    cout << "enter member " << i << endl;  
    cin >> arr[i];  
}
```

- or you could initialise each member individually

```
arr[0] = 11;  
arr[1] = 2;  
arr[2] = 6;  
arr[3] = 7;  
arr[4] = 3;
```

Constants

- constants are pieces of data which do not change throughout the program, e.g. the size of an array
- constants are defined above the main body of the program

```
#include <iostream.h>

const int ARR_SIZE = 5;

int main()
{
int arr[ARR_SIZE];

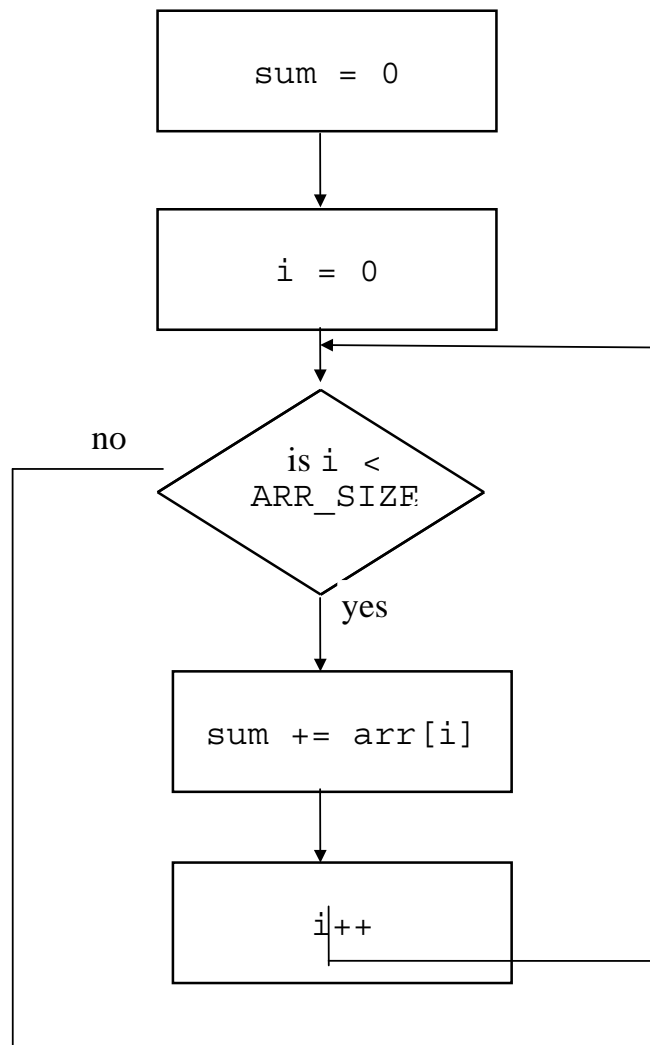
for(i=0; i<ARR_SIZE; i++)
    cin << arr[i];

.
.
.
```

- using a constant to define the array size makes the program easier to edit. If we wished to edit the program to make the array size bigger we need make only one change at the start of the program.
- it also makes the program less prone to errors. If we always refer to the array size as `ARR_SIZE` then we are less likely to accidentally give it different values in different parts of the program.
- using a constant makes the program easier to understand. When we see the word `ARR_SIZE` it is more obvious what it represents than if we simply saw the number 5
- constants are conventionally written in capital letters. This makes it easy to recognise them

Finding the Sum of an Array

- we need a variable to hold the value of the sum - let's call it `sum`
- we should initialise `sum` to 0 and then add each member of the array to `sum` in turn



- the following program fragment implements the above flowchart

```

int sum,i,arr[ARR_SIZE];

sum=0;
for(i=0; i<ARR_SIZE; i++)
    sum += arr[i];
  
```

Finding the Maximum and Minimum Values in an Array

- we will need a variable to store the maximum value. Let's call it `max`
- firstly we set `max` equal to the first member in the array and then compare `max` to each member of the array in turn. If `max` is less than the next member in the array we set `max` equal to that member

```
int max,i,arr[ARR_SIZE];  
  
max = arr[0];  
for(i=0; i<ARR_SIZE; i++)  
    if(max < arr[i])max=arr[i];
```

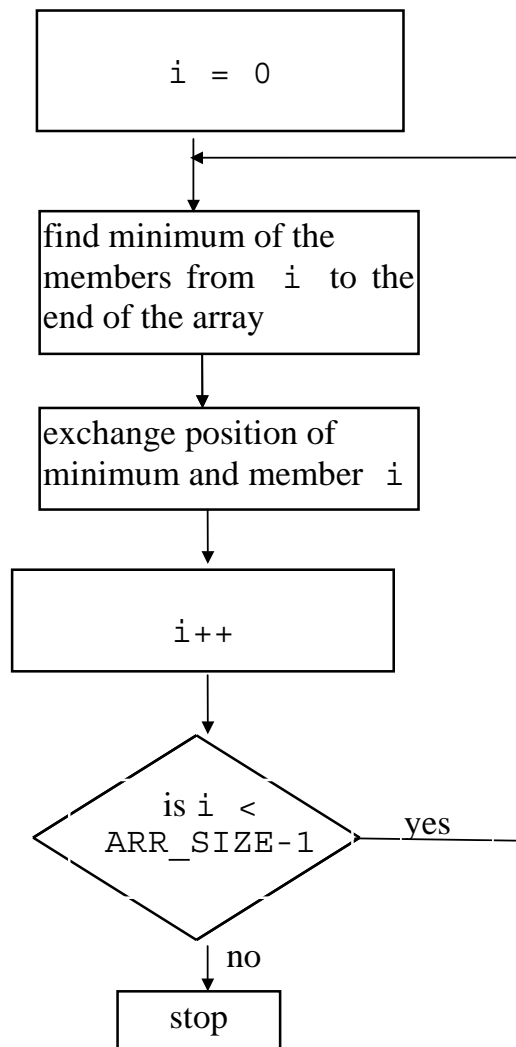
- after it has been compared with every member in the array `max` will be equal to the maximum member of the array
- it is easy to adapt the above loop to find the minimum. We now use a variable called `min` to store the minimum value

```
int min,i,arr[ARR_SIZE];  
  
min = arr[0];  
for(i=0; i<ARR_SIZE; i++)  
    if(min > arr[i])min=arr[i];
```

- note that we now use the greater than symbol `>` rather than the less than symbol `<` in the final line

Sorting an Array

- suppose you had an array of numbers in some random order and you wanted to arrange them in order of size - starting with the minimum and ending with the maximum. This is known as “sorting”.
- on the previous slide we saw how to find the minimum and maximum members of an array. We can easily adapt that code in order to sort the array.
- the basic technique is to find the member with the minimum value and then exchange its position with that of the first member in the array. Then find the minimum of the remaining members and exchange its position with the second element in the array. Then repeat the process until the members are all in order.



- here is an example using an array with 5 members

- Suppose the original ordering is

4 10 2 1 6

- The first step is to find the minimum of all 5 members. This is 1.
- Then we exchange the position of the minimum with the *first* member of the array

1 10 2 4 6

- Then we find the minimum of the remaining 4 members. This is 2.
- Then we exchange the position of this new minimum with the *second* member.

1 2 10 4 6

- Then we find the minimum of the remaining three members. This is 4.
- We exchange this with the *third* element.

1 2 4 10 6

- Finally we find the minimum of the remaining two members. This is 6.
- We exchange this with the *fourth* member.

1 2 4 6 10

And the array is now in numerical order.

- There are many other algorithms for sorting arrays. This algorithm is not the most efficient but it is the easiest to understand and program. It is called a “selection” sort.

We can express the above algorithm in code as follows

```
#include <iostream.h>
const int ARR_SIZE = 5;

int main()
{
    int i,j,imin,min;
    int arr[ARR_SIZE] = {4,10,2,1,6};

    for(i=0; i<ARR_SIZE-1; i++)
    {
        min=arr[i]; imin=i;
        for(j=i; j<ARR_SIZE; j++)
            if(min > arr[j]){min=arr[j]; imin=j;}
        arr[imin]=arr[i];
        arr[i]=min;
    }
    for(i=0; i<ARR_SIZE; i++)
        cout << arr[i] << " ";

        cout << endl;

    return 0;
}
```

- `imin` stores the position of the minimum at each iteration

- the three lines

```
min=arr[i]; imin=i;
for(j=i; j<ARR_SIZE; j++)
    if(arr[j] < min){min=arr[j]; imin=j;}
```

find the minimum of the remaining members of the array

- the two lines

```
arr[imin]=arr[i];
arr[i]=min;
```

exchange the positions of the minimum with member `i`

- this is the most difficult and complex program you have met so far. Take your time to understand it. Try it out with different arrays.

Two Dimensional Arrays

- we can store two-dimensional data in two dimensional arrays
- two-dimensional arrays are defined as follows

```
int arr[5][5];
```

This would define an array `arr` with 5 rows and 5 columns

- two-dimensional arrays are like matrices in mathematics. The members are arranged in rows and columns

```
arr[0][0] arr[0][1] arr[0][2] arr[0][3] arr[0][4]
arr[1][0] arr[1][1] arr[1][2] arr[1][3] arr[1][4]
arr[2][0] arr[2][1] arr[2][2] arr[2][3] arr[2][4]
arr[3][0] arr[3][1] arr[3][2] arr[3][3] arr[3][4]
arr[4][0] arr[4][1] arr[4][2] arr[4][3] arr[4][4]
```

- the array has two indices. The value of the first index tells you which row the member is in. The value of the second index tells you which column the member is in
- the numbers of rows and columns can be different from each other

```
int arr2[10][5];
```

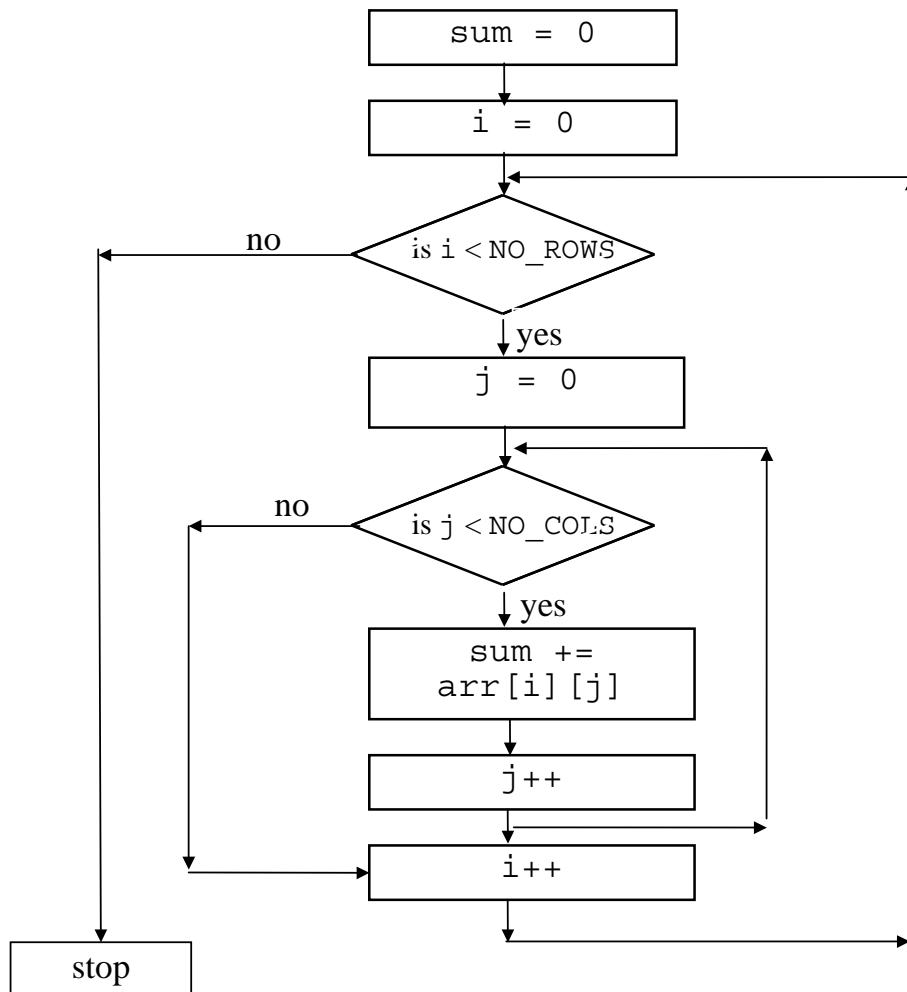
would define an array with 10 rows and 5 columns

Finding the Sum of a Two-dimensional Array

- the code for finding the sum of a two-dimensional array is very similar to that for a one-dimensional array. Here i and j represent respectively the row and column of each member of the array

```
int sum, i, j;  
int arr[NO_ROWS][NO_COLS];  
sum=0;  
for(i=0; i<NO_ROWS; i++)  
{  
    for(j=0; j<NO_COLS; j++)  
        sum += arr[i][j];  
}
```

- we now need two `for` loops: one is nested inside the other. The inner loop adds the members of row i to `sum`. The outer loop increments i by one every time the inner loop reaches the end of a row



Matrix Multiplication

- If A is an $m \times n$ matrix and
- B is an $n \times p$ matrix then
- the product of A and B is the $m \times p$ matrix AB where:

$$(AB)_{ij} = \sum_{k=1}^n A_{ik} B_{kj}$$

- for each pair i and j where:
- $1 \leq i \leq m$
- $1 \leq j \leq p$
- Let's try to translate this into C++...