Arrays

```
/*To set the elements of an array to the letters A-Z */
#define LENGTH 26
char characters[LENGTH];
int i;

for (i=0;i<LENGTH;i++)
        characters[i]= (char) ( i + (int) 'A');
```

Arrays

- In C++, there is no run-time check to make sure that the array bound is valid, so a write to element characters[30] would be attempted, perhaps causing corruption of some other data
- C++ assumes you know what you're doing!
- The result of accessing an array element outside the array bounds is undefined
- Note: arrays always start at element 0, so an array of size 26 has elements 0 to 25

- A one-dimensional array is a vector of like elements, a particular element being selected by means of the subscripting operator [] e.g.

```
int vec[4];
for(int i=0; i<4; i++ ) vec[i]=i;
```

0	1	2	3

- A two dimensional array is a vector of vectors of like elements.

```
int table[3][4];
for(int i=0; i<3; i++ )
   for(int j=0; j<4; j++ )
      table[i][j]=i*10+j;
```

00	01	02	03
10	11	12	13
20	21	22	23

A three dimensional array is a vector of two-dimensional arrays of like elements, and so on for higher dimensions.

Passing arrays as parameters to a function

- C++ arrays are not self-describing.
- This means that when an array is passed as a parameter to a function, its bounds must be described.
- As the description of the bounds of an array is set at compile time, and remains constant, this can limit the generality of the function
- I.e. It would be difficult to write a function which could handle any size of array.
- One way round the problem is to use pointers

- As the bounds of an array are not checked at run-time, the bound of the first subscript may be omitted
- The compiler is able to work out the position of the elements of the array from the bounds of the other subscripts.
- Arrays in C++ are mostly indistinguishable from pointers parameters, so they seem to be passed to the called function by reference.
- I.e. rather than passing the whole array, a reference to the array is passed
- This also means that changes to the array within the function remain changed on return.

Example: 1D array

```
#define ROW 3
void print(int arr[]);
void main(){
int numbers[ROW];
for(int i=0; i<ROW; i++)
      numbers[i] =i;
print(numbers);
}

void print(int arr[]){
   for (int i=0;i<ROW;i++)
  cout<<arr[i]<<"\t";
}
```

Example: 1D array (using a pointer)

```
#define ROW 3
void print(int *arr);
void main(){
int numbers[ROW];
for(int i=0; i<ROW; i++)
      numbers[i] =i;
print(numbers);
}

void print(int *arr){
   for (int i=0;i<ROW;i++)
  cout<<arr[i]<<"\t";
}
```

This is equivalent, but does not extend into 2D

Example: 2D array

```
const int ROW = 3;
const int COLUMN = 4;

void print(const int table[][COLUMN]);

void main(){
int numbers[ROW][COLUMN];

for(int i=0; i<ROW; i++)
  for(int j=0; j<COLUMN; j++)
      numbers[i][j]=i*10+j;

print(numbers);
}
```

```
void print(const int table[][COLUMN] )
{
   for (int i=0;i<ROW;i++){
      for (int j=0;j<COLUMN;j++)
          cout << table[i][j] << "\t";
      cout << "\n"
      }
}
```

A Pointer to an Array

- You can generate a pointer to the first element of an array simply by specifying its name, without any index. E.g.

```
int *ptr;
int sample[10];
ptr = sample;
```

- This assigns to pointer ptr the address of the first element of sample.
- In fact, in C++ there is no strong distinction between arrays and pointers (or at least single dimensional arrays)
- You could also specify the address of the first element of an array by using the & operator. E.g.

  `sample` and

  `&sample[0]` both produce the same results.

- We have seen that you cannot pass an entire array by value as an argument to a function.. This would be much too inefficient.
- Therefore, we pass a reference to the array, i.e. a pointer to the first element of the array.
- The array's name, without an index, is a pointer to the first element of the array. E.g.

```
void Display(int x[10]); //function prototype
.......
int name[10];             //Usage
Display(name);
```

Strings

- The most common usage of one-dimensional arrays is the character string.
- A string is defined as a character array that is terminated by a null, or **'\0'.**
- Therefore, you must declare character arrays to be one character longer than the largest string that they are to hold. E.g. a string of 10 characters will be:

```
char s[11];
```

- Although C++ does not have a string data type, it still allows string constants.

Strings

- A string constant is a list of characters enclosed in double quotes, e.g.: “hello there”
- Note: A character constant is a single charater enclosed in single quotes: e.g. ‘a’ or ‘9’ or ‘\n‘
- It is not necessary to manually add the null onto the end of string constants, the compiler does this automatically.
- You can assign a string constant to a character array when it is declared.

Strings

- You may not assign characters subsequently to a character array more than one at a time, i.e. you may not do this:

```
char str[10] = "Mary";   //This is allowed
cout << str;             //So is this
str = "John";            //This is not allowed
cout << str;
```

```
#include <iostream.h>
#include <string.h>
const char name[10] = "Goofy";

void main()
{
char ch = '\n';
char str[80] = "Hello there: ";

cout << str <<name<<ch;

strcpy(str,"Mary had a little lamb");   //copies string into array
cout << str <<ch;
}
```

Output:

```
Hello there: Goofy
Mary had a little lamb
```

- This will work, but we are using a C library: string.lib

Streams

- The C++ input/output system operates through **streams.**
- A stream is a logical device that either produces or consumes information.
- A stream is linked to a physical device by the C++ I/O system
- For example, the cin stream is linked by default to the keyboard, the cout stream is linked to the screen.
- All streams behave in the same way even though the actual physical devices they are connected to may differ substantially.
- Because of this, the same C++ I/O functions can operate on virtually any type of physical device.

Array-Based I/O

- This stream-based I/O allows array-based I/O also.
- In this case, the RAM of the computer is the input device, the output device, or both.
- To use array-based I/O in your programs you must include strstream.h
- The array-based I/O classes are
  - istrstream: to create input streams
  - ostrstream: to create output streams
  - strstream: to create input/output streams
- To link an output stream to an array, you use the ostrstream constructor, which takes a pointer to a character array to output to, and the length of that array, as parameters.
- Now for the previous program, in C++:

```
#include <strstrea.h>
#include <iostream.h>

const char name[10] = "Goofy";

void main()
{
char ch = '\n';
char str[80] = "Hello there: ";
ostrstream outs(str, sizeof(str));

cout << str <<name<<ch;

outs << "Mary had a little lamb";
cout << str <<ch;
}
```

Using an array as input

- To link an input stream to an array, use the istrstream constructor.
- This takes a pointer to a character array that will be used as a source of characters each time input is performed on the stream.
- If you want only part of a string to be used for input, you also pass in the size of the sub-array you want to use.
- Here, only the specified number of elements will be used.

```
#include <strstream>
#include <iostream.h>
void main()
{
int i;
char str[80];
float f;
char inputArray[] = "10 Green Bottles 0x88 12.23 Done";
istrstream ins(inputArray);
ins>> i;            //Read in: 10 Green
ins>> str;
cout << i << " "<< str<< " ";
ins >> str;         //read in bottles 0x88 12.23
ins >> i;
ins >> f;
cout << str <<" "<< hex << i << " "<<f ;
ins >> str;         //read in done
cout << " "<<str;
}
```

Input/Output Array-Based Streams

- To create an array-based stream that can perform both input and output, use the strstream constructor function.
- You pass it a pointer to the character array to be used for both input and output, the size of the array, and the mode, which will be ios::in|ios::out. E.g:

```
#include <strstrea.h>
#include <iostream.h>
void main()
{
int i;
char buffer[80];
strstream ioStr(buffer, sizeof(buffer), ios::in|ios::out);

int a,b;
ioStr << "10 20 testing";
ioStr >> a >> b>> buffer;

cout << a << " " << b << " " << buffer << "\n";
}
```