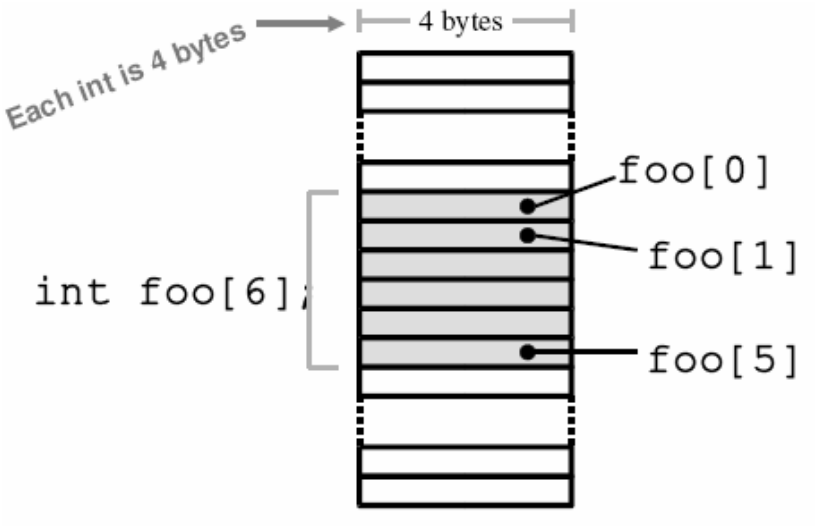
C++ Arrays

- An array is a consecutive group of memory locations.
- Each *group* is called an element of the array.
- The contents of each element are of the same *type*.
  - For example, could be an array of int, double, char, …
- We can refer to individual *elements* by giving the position number (index) of the element in the array.



C++ Arrays start at 0 !!!!!!!

- The first element is the 0th element!
- If you declare an array of *n* elements, the last one is number *n-1*.
- If you try to access element number *n* it is an error!

Array Subscripts

- The element numbers are called subscripts.
- For example: In foo[i], foo is the name of the array, and "i" is the subscript.
- A subscript can be any integer expression:

  These are all valid subscripts:

  foo[17] foo[i+3] foo[a+b+c]

```
//example 10.cpp
	#include <iostream>
	int main(void) {
		int facs[10];  //declare an int array of size 10
```

```
        for (int i=0;i<10;i++)   facs[i] = i;          //initialize
        for (int i=0;i<10;i++)
                cout << "facs[" << i << "] is " << facs[i] << endl;
}
```

Initialization

- You can initialize an array when you declare it (just like with variables):

```
int foo[5] = { 1,8,3,6,12};          //size of array is 5.
double d[2] = { 0.707, 0.707};
char s[] = { 'R', 'P', 'I' };        //size of array is automatically determined
                                     //from the size of elements
```

```
//example11.cc
#include <iostream>

int main()
{
 int    ar1[100];   // 100 elements, 0 to 99
 float  ar2[4] = { 1.01, 2.02, 3.03, 4.04 };
 int    i;

 cout << "size of ar2 " << sizeof(ar2)<< endl;
 for (i = 0; i <= 99; ar1[i] = i, i++);
 for (i = 0; i < 4; cout << ar2[i] << endl, i++);
 return 0;
}
```

This example shows us quite a bit about arrays:

- The array *ar1* has 100 elements indexed from 0 to 99
- Array *ar2* has 4 elements and each element is initialized to contain some value, eg *ar2[0]* contains *1.01*.
- The index for ar2 could have been omitted, we could have written *float ar2[] = { 1.01, 2.02, 3.03, 4.04 };*.
  The size of *ar2* would be determined at compile time by the number of initialization values.
- The *sizeof()* function is a standard function which returns the size, in bytes, of its argument.
- The second for loop uses the comma operator:
  ar2[i] << endl, i++

We can declare arrays of any data type and the size of an array is only limited by the available memory. Remember that each element of an array is the same kind.

```
 // example12.cc
// Program reads numbers into an array
// and prints them out in reverse order.

#include <iostream>
using namespace std;

const int MAX = 10;

int main ()
{
   int  numbers[MAX];
   int  index;

   for (index = 0; index < MAX; index++)
   {
      cout << "Please enter an integer number: " << endl;
      // store value into numbers array
         cin >> numbers[index];
   }

   for (index = MAX - 1; index >= 0; index--)
      // Print numbers on the screen
         cout << "number[" << index << "] = " << numbers[index] << endl;
   return 0;
}


//example13.cpp
// This program manipulates values in an array.

#include <iostream>
using namespace std;

int main ()
{
    const int MAX_ARRAY = 5;
    int  numbers[MAX_ARRAY];
    int  index;
    int  sum;

   // Stored values in the array.
   for (index = 0; index < MAX_ARRAY; index++)
      numbers[index] = index * index;
```

```
   // The values in the array are summed.
   sum = 0;
   for (index = 0; index < MAX_ARRAY; index++)
      sum = sum + numbers[index];
   cout  << "Sum is "  << sum  << endl;

   //  print out the values of the array
   for (index = 0; index < MAX_ARRAY; index++)

   cout << "Array elements are " << numbers[index] << endl;


   return 0;
}


//example14.cpp
/**************************************************************************
**
Filename:            function.cpp
Purpose: This C++ program is to create a one-dimensional array which can
            hold 10 numbers. Fill the array with integer values.
            Then print out the array and print out the largest value and
            smallest value in the array.
**************************************************************************
**/

#include <iostream>
using namespace std;

int getMax(int[], int);

//add function prototype of function getMin()

int main()
{
  int array[10], max, min;
  int i;
  for (i=0; i<10; i++)
  {
    cout << "Enter an integer value: " ;
    cin >> array[i];
    cout << endl;
  }
  max = getMax(array, 10);
```

```
  //add code to call the function getMin() and assign the value to varible min

  cout << "The array is: " << endl;
  for (i=0; i<10; i++){
    cout << array[i] << endl;
  }
  cout << "The largest value is: " << max << endl;
  cout << "The smallest value is: " << min << endl;
  return 0;
}

//*********************************************
int getMax(int ary[], int len)
{
  int max = ary[0];
  int i;
  for (i=0; i<len; i++)
  {
    if (max<ary[i])
       max = ary[i];
  }
  return max;
}

//*********************************************
//write the function  getMin(int ary[], int len)
```

Character Arrays - C style strings

This topic is included to show you that character arrays can be used in C++. However when using character arrays as strings they can be quite tricky to deal with.

Later I introduce a string class which is available in C++. This provides a much nicer way to handle strings. For now just accept that we can deal with arrays of characters if we want to.

```
//example12.cc
#include <iostream>

int main(){
  char string1[ ] = {'F','r','e','d','\0'};
  char string2[ ] = {"Fred"};
     char string3[] = “RPI without PI is like meat without eat”;
```

```
  cout << "Size of string1 is " << sizeof(string1)
       << " bytes" << endl;
  cout << "string1 contains " << string1 << endl;
  cout << "Size of string2 is "
       << sizeof(string2) << endl;
  cout << "string2 contains " << string2 << endl;
  cout << "Size of string3 is "
       << sizeof(string3) << endl;
  cout << "string3 contains " << string3 << endl;

  return 0;
}
```

```
The output is:
```

Size of string1 is 5 bytes
string1 contains Fred
Size of string2 is 5
string2 contains Fred
Size of string3 is 39
String3 contains RPI without PI is like meat without eat

- You can see that neither *string1* nor *string2* have a size declared. The size is determined by the initialisation.
- A C style string is terminated by the *null* character which is written '\0'. This is essential when using C style strings.
- A string constant like *"Fred"* doesn't require the null character, the compiler will take care of its addition.
- The size of the C style array is always the length of the string plus 1 for the null terminating character.

I will reiterate the last point:

With this null-terminated string type the length of the variable need not be specified, the compiler will automatically determine the size according to the placement of null. The null character takes up a place in the array so whenever character arrays are being declared make sure you allow for the null and be sure that any string values you use are terminated with the null character. Programs can do frightening things to your computer system when strings are improperly terminated.

Later I will show you much better ways to deal with strings using the string class in C++.

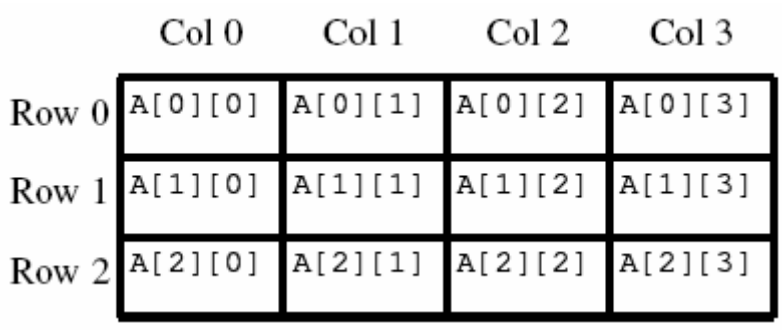
Multi-dimensioned Arrays

We are not limited to single dimensioned arrays; we can have as many dimensions to an array as we need:

You can create an array of arrays:

```
int a[3][4];
```

For example, above example creates the following array:



We can initialize the entries of the following array as follows:

```
for (int i=0;i<3;i++)
        for (int j=0;j<4;j++)

                a[i][j] = i+j;
```

Example:

```
const int NumStudents = 10;
const int NumHW = 3;
double grades[NumStudents][NumHW];

for (int i=0;i<NumStudents;i++) {
        for (int j=0;j<NumHW;j++) {
                cout << "Enter HW " << j <<" Grade for student number " <<i << endl;
                cin >> grades[i][j];
        }
}
```

When passing 2-D or multi-dimensional arrays to functions, you do not need to specify first dimension, but you need all the others.

```
double student_average( double g[][NumHW], int stu) {
        double sum = 0.0;
        for (int i=0;i<NumHW;i++)
                sum += g[stu][i];
        return(sum/NumHW);
}
```

```
//ex21.cpp
#include <iostream>
#include <iomanip>

int main()

{ int num[10][10];
  int row, column;

  for (row = 0; row < 10; row++)
    for (column = 0; column < 10; column++)
      num[row][column] = row * column;
  for (row = 0; row < 10; row++)
    for (column = 0; column < 10; column++)
      cout << setw(4) << num[row][column] << "  ";

  return 0;
}
```

The declaration int num[10][10] declares a two-dimensioned array of size 10 by 10. Any reference to an element in the array then uses a pair of indices or subscripts, eg

num[5][7]

which refers to the 6th row, 8th column of the array. Since in C arrays are zero-based the first element is [0][0]. Take care when referencing array elements, C doesn't check subscript boundaries.

```
   0   0   0   0   0   0   0   0   0   0
   0   1   2   3   4   5   6   7   8   9
   0   2   4   6   8  10  12  14  16  18
   0   3   6   9  12  15  18  21  24  27
   0   4   8  12  16  20  24  28  32  36
   0   5  10  15  20  25  30  35  40  45
   0   6  12  18  24  30  36  42  48  54
   0   7  14  21  28  35  42  49  56  63
   0   8  16  24  32  40  48  56  64  72
   0   9  18  27  36  45  54  63  72  81
```

The output from example 3 is shown here.

The neat spacing is provided by the stream manipulator *setw(4)*. This manipulator sets the output field width to 4 characters

Arrays and Passed by Reference

- Arrays are always passed by reference
- Inside a function any changes you make to array values are for keeps!
- You can write functions that modify the contents of an array.
- You need to make sure that a function knows how big the array is!!!

Pay attention to size problem

```
int read_array( int a[] ) {
        int i=0;
        int val;
        do {
                cout << "Enter next value, 0 to end\n";
                cin >> val;
                if (val) a[i++] = val;
        } while (val);
        return(i); // returns the number or numbers
}
```

The problem is that the function might go beyond the size of the array.
Result: Segmentation Violation (or worse!).

Exercise:

1. Write a C++ function that initializes an array of int with a specific value. Prototype of the function looks like as follows:
   ```
   void init_array(int a[], int len);
   ```
2. Write a function that adds the elements of two different arrays and stores the results in a third array (all arrays are the same size).
   ```
   // c = a + b
   void add_arrays(int a[], int b[], int c[], int len);
   ```

3. Write a function that returns true of false: true only if a value x is found in an array a
   ```
   int search_array(int a[], int len, int x);
   ```