Arrays

Definition:

An array is a collection of variables of the same type that are referenced by a common name.
-In C++, all arrays consist of contiguous memory locations.
-To refer to an element or particular location in the array, we specify the name of the array and the position number –or index or subscript- of that element.
Arrays are somewhat like a filing cabinet; they use an indexing system to find the elements in the cabinet.

Example of an array named a of 5 elements

Element	Value	Address in memory
a[0]	-45	1000
a[1]	6	1001
a[2]	0	1002
a[3]	72	1003
a[4]	43	1004

- What is the value of the a[4] ?
- What is the value of a[1] ?
- What is the index and value of the last element of the array a?
- What is the subscript and value of the first element of the array a?
- How would we get the value of adding the first element and the last one? a[0]+a[4]
- What are the lowest index (lower boundary) and the highest index (higher boundary) of an array of 10 elements?

Any element in an array can be accessed by using:

- the name of the array ,
- followed by [,
- then its position in the array,
- followed by].

In C++, all indexes start at zero.
The first element of the array a is referred to as a[0], the second element as a[1], the third as a[?]..

The position of an element is called its index

- What is the value of the a[4] ?

- What is the value of a[1] ?
- What is the index and value of the last element of the array a?
- What is the subscript and value of the first element of the array a?
- How would we get the value of adding the first element and the last one? a[0]+a[4]
- What are the lowest index (lower boundary) and the highest index (higher boundary) of an array of 10 elements?

Array Declaration:

Like any other variable in C++, arrays must be declared. The general format for declaring an array is:

datatype array_name[size];

Where
data_type is one of C++ data types: int, char, float, double, bool
array_name is a C identifier
size defines how many elements the array will hold.
For example to declare a 100-element array called balance of type double, we use this statement:
double balance[100];

What are we declaring in the following?
char p[10];

int sample [12];

We can declare more than one array in a single type declaration
int id[4], index[6];

Assigning Values to Array elements:

To assign a value to an element of an array, we use:

p[i] = expression;

Where *p* is the name of the array, and *i* is the index or position of the element in which we want to assign the value; *expression* can be any valid C++ expression.

To assign the value of an array element into a variable, we use:

var = p[i];

Example of use of an array:

```
/*  Introducing array's  */
     #include <iostream>
     using namespace std;

     int main(void)
     {
     int numbers[100];
     numbers[3] = 10;
     --numbers[3];
     cout<< “The 4th element of array numbers is"<<
     numbers[3];
     }
```

Sample Program Output

```
The 4th element of array numbers is 9
```

This program declares one array of type integer with a 100 elements.

It then assigns 10 to the 4th element of the array at index 3, decrements it by 1 and prints out its value.

Initializing an Array with a declaration

An array can be initialized by using the following format:

array_name[size] ={value_at_index0, value_at_index1…. value_at_indexsize-1}

Example:

```
#include <iostream>
using namespace std;
     int main(void){
     int x;
     static int val[9] = { 1,2,3,4,5,6,7,8,9 };
     static char word[] = {'H','e','l','l','o'};
     for( x = 0; x < 9; ++x )
```

```
    cout<< "Values “<<x <<” is “<< val[x]<< “\n";
  return 0;}
```

Sample Program Output

```
Values[0] is 1
Values[1] is 2
....
Values[8] is 9
```

The previous program declares two arrays, *val* and *word*. The array named *val* has been declared with a size of 9. So *val* consist of 9 elements (indexed 0 to 8)

For the array *word*, there is no number or variable inside the square brackets to indicate how large the array is.

In this case, C initializes the array to the number of elements that appear within the initialization braces so the char array *word* has 5 elements indexed 0 to 4.

-If there are fewer initializers or values than elements in the array, the remaining elements are automatically initialized to 0.

```
#include <iostream>

using namespace std;

    int main(void){

    int x;

    int  val[9] = { 1,2,3,4,5,6,7 };

    static char  word[] = {'H','e','l','l','o' };

    for( x = 0; x < 9; ++x )

    cout<<"Values ”<<x <<”  is  “<< val[x] <<endl;

    return 0; }
```

Sample Program Output

```
Values[0] is 1
Values[1] is 2
....
Values[6] is 7
Values[7] is 0
Values[8] is 0
```

-If there are more initializers than elements in the array, this can cause a syntax error.

```
#include <iostream>

    using namespace std;

    int main(void)

    {

    int x;

    int  val[9] = { 1,2,3,4,5,6,7, 8, 9, 10 };

    static char  word[] = {'H','e','l','l','o' };

    for( x = 0; x < 9; ++x )

     cout<< "Values  “<< x << “ is “<<  val[x]<<endl;

    return 0;

    }
```

Error E2225 array-t.cpp :too many initializers in function main()

Initializing an Array using a loop:

The following program shows how to initialize all the elements of an integer based array to the value 10, using a for loop to cycle through each element in turn.

```
#include <stdio.h>
int main(void)
{
int count;
int  ex[100];
for( count = 0; count < 100; count++ )
     ex[count] = 10;
return 0;
}
```

The loop assigns 10 to ex[count] until countbecomes 100, at which point it terminates and the array elements are all initialized to 10.

Using a constant to define the array size:

```
#include <iostream>

using namespace std;

const  int SIZE = 10;

int main(void){

int  s[SIZE], j;

for( j = 0; j < SIZE; j++ )

    s[j] = 2+ 2*j;

cout<<  "Element\tValue \n";

for( j = 0; j < SIZE; j++ )

    cout<< j<<"\t"<< s[j]<<endl;

return 0;
```

}Sample Program Output:

```
Element          Value

  0                    2
  1                    4
  2                    6
  3                    8
  4                    10
  5                    12
  6                    14
  7                    16
  8                    18
  9                    20
```

The program initializes the elements of a 10 element array, to the values 2, 4, 6,…,20 and prints it in a tabular format.
The values are generated by multiplying the loop counter by 2 and adding 2.

The program below sums the values contained in a twelve-element integer array.

```
/*compute the sum of the elements of the array */

#include <iostream>

using namespace std;

const int SIZE= 12;

int main() {

int a[SIZE]= { 1, 3, 5, 4, 7,2, 99,16,45,67,89,45}, i,
total =0;

for (i=0; i<SIZE; i++)

    total +=a[i];

cout<<" Total of array element values is" <<
total<<endl;

return 0;

}
```

Sample program output:

```
Total of array element values is 383
```

Using loops for sequential access:

Very often, the elements of an array have to be processed in a sequence, starting with element zero. In C++, we can accomplish this processing easily using an indexed for loop, a counting loop whose loop control variable runs from 0 to one less the array size.

Using the loop counter as an array index gives access to each array element in turn

For example: The following array square will be used to store the squares of the integers 0 through 10 (e.g., square[0] is 0, square[10] is 100). We assume that SIZE has been defined to be 11.

```
 int square[SIZE], k;

for(k=0;  k<SIZE;       ++k)

    square[k] = k*k;
```

This will initialize the array square as follows

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
0	1	4	9	16	25	36	49	64	81	100

Problem Statement:

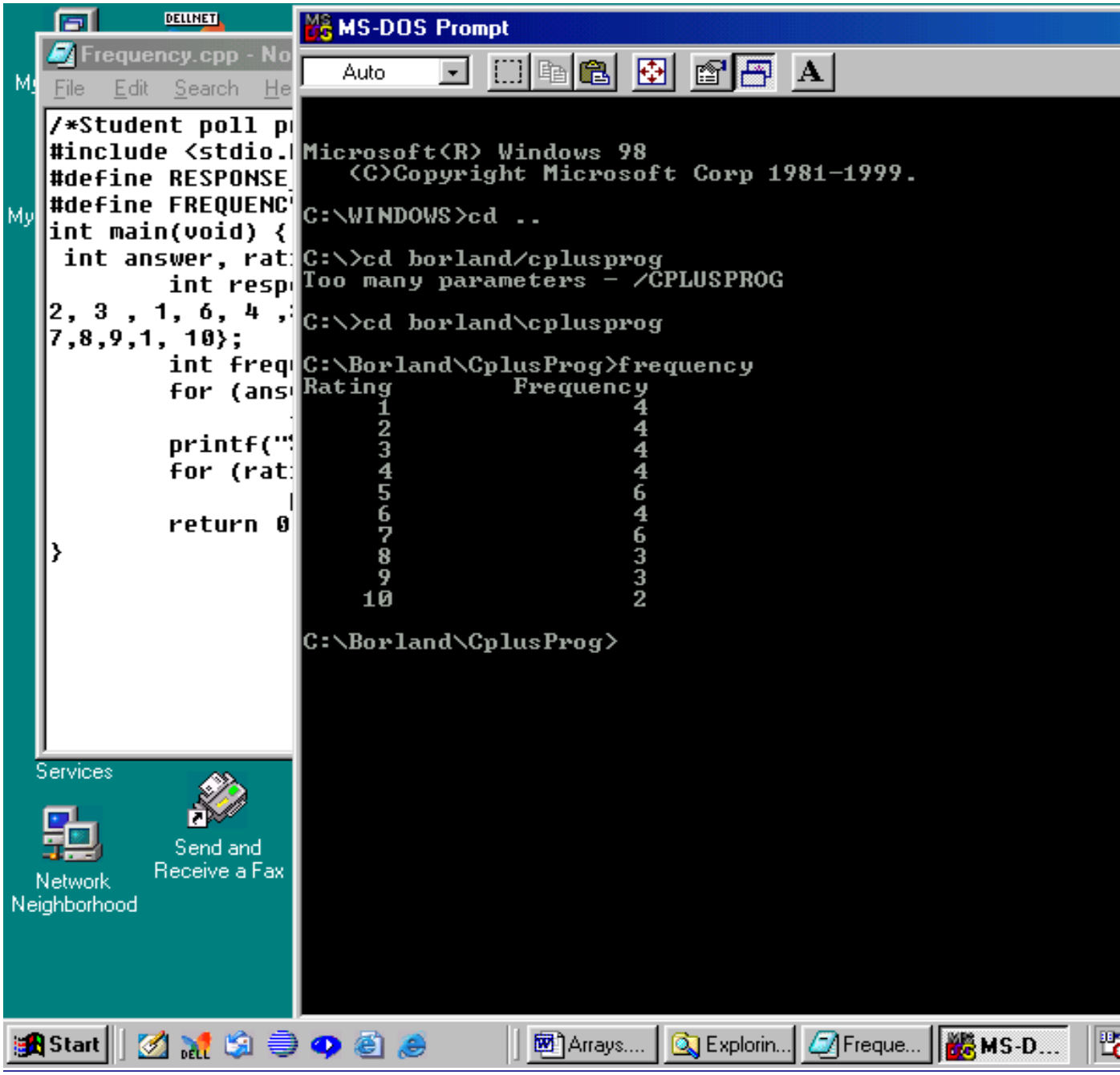
Forty students were asked to rate the quality of the food in the cafeteria from 1 to 10. (1 means awful, 10 means excellent). Place the forty responses in an integer array and summarize the results of the poll. We wish to summarize the number of responses of each type.

We use two arrays:

1. An array response of 40 elements that has the students' responses.
2. An array frequency of 11 elements to count the number of occurrences of each response. The element 0 is ignored. So that frequency[1] holds the frequency of the answer 1.

C++ program:

```
/*Student poll program */

#include <iostream>

using namespace std;

const int RESPONSE_SIZE = 40;

const int FREQUENCY_SIZE= 11;

int main(void) {

 int answer, rating;

int response[RESPONSE_SIZE]= {1, 2, 5, 6, 7 , 4, 5, 7 ,9, 10, 2, 3 , 1, 6, 4 ,3 ,5,6,
7,2, 5 ,9, 8,3, 3, 5, 4,2, 1, 7, 7,8, 4, 5, 6, 7,8,9,1, 10};

int frequency[FREQUENCY_SIZE] = {0};

for (answer = 0; answer <= RESPONSE_SIZE -1; answer++)

        ++frequency[response[answer]];

        cout<<  "Rating\t  Frequency";

for (rating =1; rating <=FREQUENCY_SIZE-1; rating++)

        cout<<  rating<<"\t"<< frequency[rating];

        return 0;}
```


DELLNET
Frequency.cpp - No
File Edit Search He
/*Student poll p
#include <stdio.
#define RESPONSE
#define FREQUENC
int main(void) {
int answer, rat
int resp
2, 3 , 1, 6, 4 ,
7,8,9,1, 10};
int freq
for (ans
printf("
for (rat
return 0
}
Services
Network Neighborhood
Send and Receive a Fax
MS-DOS Prompt
Auto
Microsoft(R) Windows 98
(C)Copyright Microsoft Corp 1981-1999.
C:\WINDOWS>cd ..
C:\>cd borland/cplusprog
Too many parameters - /CPLUSPROG
C:\>cd borland\cplusprog
C:\Borland\CplusProg>frequency
Rating Frequency
1 4
2 4
3 4
4 4
5 6
6 4
7 6
8 3
9 3
10 2
C:\Borland\CplusProg>
Start
Arrays....
Explorin...
Freque...
MS-D...

```
for (answer = 0; answer <= RESPONSE_SIZE -1; answer++)
    ++frequency[response[answer]];
```

This for loop takes the responses one at a time from the array response whose values are between 1 and 10.

It increments one of the ten frequency counters in the array frequency.

For example:

for answer =0, response[0] = 1,

++frequency[response[0]]= ++frequency[1]= 1

For answer = 1, response[1] =2,

++frequency[response[1]]= ++frequency[2] =1

….

answer= 10, response[10]= 2

++frequency[response[10]]= ++frequency[2] =2, which increments the array element 2 of the array frequency.

Note: Regardless of the number of students surveyed, the responses are always between 1 and 10 so only an eleven-element array is required to summarize the results.

Character Arrays or String:

The most common use of one-dimensional arrays is as character strings.

In C++, a string is defined as an array of individual characters.

For example, the string “hello” is stored as a character array.

Initializing a character array:

Strings or character arrays can be initialized as:

char hello[] = “Hello”;

This initializes the elements of the array hello to the individual characters in the string literal “Hello”.

The size of the array is determined by the compiler based on the number of characters in the string, plus a special character **\0** that the compiler adds to signify the end of the string.

The array is terminated by **null.**

Null is specified as the ‘\0’ character .

When we declare the size of a character array, we need to declare the array to be one character longer than the largest string that it can hold.

For example, to declare the hello array that holds a 5-character string, we would write:

char hello[6];

This makes room for the **null** at the end of the string.

If we were to declare the array by enumerating its elements, we would do it as:

char hello[6]= {‘H’, ‘e’, ‘l’, ‘l’, ‘o’, ‘\0’};

We can access individual characters in a string directly by using subscript notation, for example, hello[0] is the character ‘H’, and hello[4] is the character ‘o’ etc..

To input a string into a character array we use cin. For example:

char string2[20];

Creates a character array capable of storing 19 characters and a terminating null character.

cin>> string2;

Reads a string from the keyboard into a character array string2.

Inputting and outputting strings:

```
#include <iostream>
using namespace std;
    int main(void) {
    char string1[20], string2[]=”this is a string”;
    int i;
    cout<< "Enter a string: ";
    cin>>  string1;
    cout<< “String1 is  “<< string1 << “ and String2 is ”, << string2;
    for (i=0; string1[i] != '\0'; i++)
        cout<<  string1[i];
     cout<< "\n";
     return 0;
    }
```

Sample output program:

```
Enter a string: Hello
String1 is Hello and String2 is this is a string
Enter a string: abcdefghijklmnopqrstuvwxyz
String1 is abcdefghijklmnopqrstuvwxyz and String2 is uvwxyz
a b c d e f g h i j k l m n o p q r s t u v w x y z
```

Two-dimensional Arrays:

C++ supports multidimensional arrays. The simplest form of the multidimensional array is the matrix or two-dimensional array.

A two dimensional integer array d of size 10 and 20 is declared as:

```
int d[10][20];
```

Each dimension is placed in its own set of brackets. With two bracket pairs, we obtain a two-dimensional array. To obtain arrays of higher dimension, we simply add more brackets. With each bracket, another array dimension is added.

Declarations of arrays	*Remarks*
int a[100]	A one-dimensional array
int b[2][7]	A two-dimensional array
int c[5][3][2]	A three-dimensional array

The array a has 100 elements, the array b has 2×7 =14 elements and c has 5× 3× 2=30 elements.

Although, array elements are stored sequentially, it is convenient to think of a two-dimensional array as stored in a row-column matrix, where the first index indicates the row and the second indicates the column.

-The elements of a given row all share the same first index, for example, the elements in the first row all have the index zero.

The elements of a given column all share the same second index.

int b[2][3]

	col 1	**col 2**	**col 3**
row 1	b[0][0]	b[0][1]	b[0][2]
row2	b[1][0]	b[1][1]	b[1][2]

-A two-dimensional array is initialized at declaration time as such, (example for an array

int b[2][3]= {{1, 2, 3}, {4, 5,6}};

The values are grouped by row in braces, and

{1, 2, 3} initializes the elements b[0][0], b[0][1], b[0][2]

{4, 5,6} initializes the elements b[1][0], b[1][1], b[1][2]

	col 1	**col 2**	**col 3**
row 1	1	2	3
row2	4	5	6

Let's write a program that fills a two dimensional array. Each value in the array is the sum of its indices.

For example, the element at the position (0, 0) has the value 0, the element at the position (1,2) has the value 3, and so on. The program then prints them row-by-row:

```
#include <iostream>
using namespace std;
int main() {
    int t, i, num[3][4];
    for (t=0; t<3; t++)
        for (i=0; i<4; i++)
        num[t][i]= t+i;
        /*now print them out*/
        for (t=0; t<3; t++) {
            for (i=0; i<4; i++)
        cout<< num[t][i];
    cout<<”\n"; }    }
```

Sample program output:

```
0   1   2   3
1   2   3   4
2   3   4   5
```

	column0	column1	column2	column3
Row0	0	1	2	3
Row1	1	2	3	4
Row2	2	3	4	5

Initializing an array to 0

If an array of storage class automatic is not explicitly initialized, then array elements start with garbage values.

Static and global or external arrays are however, initialized to zero by default.

A simple way to initialize all array elements to 0:

int a [2][4][6]={0};

Passing Arrays to functions:

To pass an array a of dimension n as an argument to a function, we use the general form:

function_name (data_type a[], int n), where: function_name is the name of the function, a is the name of the array, n is the dimension of the array.

```
#include <iostream>
using namespace std;
int maximum(int a[],int b);/*prototype */
int maximum( int values[], int a)
{
      int  max_value, i;
      max_value = values[0];
      for( i = 0; i < a; ++i )
            if( values[i] > max_value )
                  max_value = values[i];

      return max_value;
      }

int main(void){
int values[5], i, max;

cout<<”Enter 5 numbers\n");
```

```
for(i = 0; i < 5; ++i )
      cin >> values[i] ;

max = maximum( values, 5 );
cout << "\nMaximum value is “<< max<<endl;
return 0;
}

Sample Program output:
34 56 78 89 43

Maximum value is 89.
```

C++ passes arrays to functions using simulated call by reference, this means that:

- By putting the name of the array, its real address in memory is passed.
- The array elements themselves are not copied
- The function manipulates the array itself and not a copy of the array.
- Any changes made on the elements of the array by the function will update the array.

To pass an element of an array to a function we use its indices and it behaves like any other variable.
The prototype of a function, which takes an array argument, is:
return_type function_name(array_type[], int)

Passing two-dimensional arrays:

A function receiving a two-dimensional array as a parameter must define at least the length of the second dimension.
This is because the C compiler needs to know the length of each row for it to index the array correctly.
For example, a function that receives a 2-dimensional integer array with (10, 10) is declared like this:

```
funct1(int x[] [10]) {
…….
}
```

It is not necessary to specify the first dimension, but specifying the second dimension is required.
Otherwise, the compiler cannot determine where rows start and end.

Sorting Arrays: The bubble sort:

```
#include <iostream>
using namespace std;
const int SIZE = 10;
main() {
 int a[SIZE] ={2, 6, 7, 12, 65, 87, 34, 52, 1, 5};
 int j, pass, hold;
 cout<<" Data in original order \n";
  for (j=0; j< SIZE; ++j)
      cout<< a[j] << " ";
  for (pass =1; pass <= SIZE; pass++)
      for (j=0; j<= SIZE-2; j++)
           if ( a[j] > a[j+1] ) {
                hold = a[j];
                a[j] =a [j+1];
                a[j+1] = hold;
                }
 cout<< "\nData items in ascending order \n";

 for (j=0; j<=SIZE-1; j++)
      cout<< a[j] <<" ";
cout<< "\n";
return 0;
}
```