Arrays

(Derived data type)

Arrays (Derived data type)

- An array is a group of related data items that share a common name.
- For example, we can define an array name marks to represent a set of marks of a group of students.
- i.e. int marks[20]
- Then a particular value is indicated by writing a number called index or subscript in brackets after the array name.
- For example, marks[10], represents the marks of 11th student

Arrays cont…

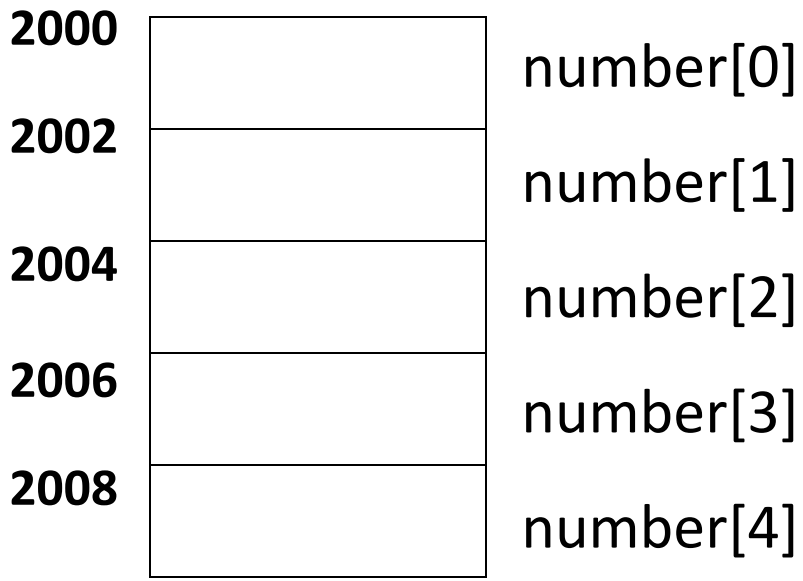
- The ability to use a single name to represent a collection of items.
- So items can be referred by specifying the item number.
- One-dimensional array
  - A list of items can be given one variable name using only one subscript and such a variable is called a one-dimensional array.
  - The subscript or index of an array begins with number 0.
  - That is x[0] is allowed. And refers to the 1st element of the array x[ ].

Arrays cont…

- If we want to represent a set of five numbers with an array variable number then first we have to declare it as:

```
int number[5];
```

- So the computer reserves five storage locations as :



Arrays cont…

- Then the value to the array elements can be assigned as:

```
number[0] = 35;
number[1] = 40;
number[2] = 20;
number[3] = 57;
number[4] = 19;
```

Arrays cont…

This would cause the array number to store the values as:

35	number[0]
40	number[1]
20	number[2]
57	number[3]
19	number[4]

Arrays cont…

- Then these elements may be used in programs just like any other C variable.
- Like: a = number[0] +10;

  number[4]=number[1] + number[2];
- The index of an array can be integer constants or any expressions that results in integer.

Declaration of array

- Like any other variable in C, array must be declared before using it.
- The general form of array declaration is:

  type variable-name[size];

- type can be - type of element contained in the array. i.e. int, float, char etc.
- size – is the maximum number of elements that can be stored inside the array.

Declaration of array Cont…

- float height[50];
- Declares the height to be an array containing 50 real elements.
- Any subscripts between 0 to 49 are valid.
- NOTE: C performs no bound checking for array and therefore, care should be taken to ensure that the array indices are within the declared limits.

Initialization of arrays

- type array_name[size]={..list of values};
- I.e. **int arr[5]={30,20,25,50,70};**
- Here if you write **int arr[5]={1,2,3};** then the remaining element will get initialize to zero.
- The size is optional only when you assign the values at the declaration time. i.e. int **arr[]={**1,2,3,4,5,6,7};
- Above instruction will create array of seven integers and assign values.
- **int arr[5] = {0}** will initialize all elements to zero

Arrays Cont…

- Suppose you are having int arr[100];
- You want to give first 50 to zero and the last 50 to one.
  - for(i=0;i<50;i++) arr[i] = 0;
  - for(i=50;i<100;i++) arr[i] = 1;

Entering Data into an Array

```
int marks[30];
for (int i = 0 ; i <= 29 ; i++ )
{
        printf(“Enter Marks:”);
        scanf(“%d”,&marks[i]);
}
```

WAP to find average marks obtained by a class of 5 students in a test

/*** 1st Method***/

```
total =marks[0]+
   marks[1]+ marks[2]+
   marks[3]+ marks[4];

res=total/5;
printf("%f", res );
```

/*** 2nd Method***/

```
for( i=0;i<5;i++)
{
total=total + marks[i];
}

res=total/5;
printf("%f", res);
```

Write a program to find average marks obtained by a class of 30 students in a test.

```
main( )
{
float res, total = 0 ;
int i ;
float marks[30] ;

 /* store data in array */
for ( i = 0 ; i <= 29 ; i++ )
{
    printf("Enter marks:" );
    scanf("%f",&marks[i]) ;
}
```

```
 /* read data from an array*/

for ( i = 0 ; i <= 29 ; i++ )
{
            total= total+ marks[i] ;
}
res = total/ 30.0 ;

printf("Average marks =%f",res);
}
```

Ex1: Program to print reverse array in C

```
for(loop = 9; loop >= 0; loop--)
        printf("%d ", array[loop]);
```

Ex2: Array concatenation program in C

Examples:

- Calculate Avg of array elements
- Store cost of 10 products and find product with the highest and lowest cost
- Program to copy an array to another array in reverse order
- Separate even and odd no.s
- Linear search

Largest & smallest of array elements

```
smallest=Arr[0];
largest=Arr[0];
for(i=1;i<n;i++)
 {
        if(Arr[i]<smallest)
                smallest=Arr[i];
        if(Arr[i]>largest)
                largest=Arr[i];
}
Printf("The highest cost is : %d", largest);
Printf("The lowest cost is : %d", smallest);
```

WAP to copy an array to another array in reverse order

```
#include<stdio.h>
#define size 5
int main(){
int a[size]={10,20,30,40,50}, b[size], i;
for(i=0;i<size;i++)
{
   b[i]=a[size-1-i];
}
for(i=0;i<size;i++)
{
   printf("%d\n",b[i]);
}
```

Even and odd numbers in array

```
int arr[20],even[20],odd[20],i,j=0,k=0;
for(i=0; i<20;i++)
{
 if(arr[i]%2==0)
{
        even[j]=arr[i];
        j++;
}
else
{
        odd[k]=arr[i];
        k++;
}
}
```

Two Dimensional Array Illustration

	0	1	2	3	4
0					
1					
2					
3					
4					

```
int matrix[5][5];
```

	0	1	2	3	4
0					
1					
2		7			
3					
4					

```
matrix[2][1] = 7;
```

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9
3	10	11	12

```
Int m[][]= {
   {1, 2, 3},
   {4, 5, 6},
   {7, 8, 9},
   {10, 11, 12}
};
```

Initialization of arrays

- type array_name[size]={..list of values};
- I.e. **int arr[5]={30,20,25,50,70};**
- Here if you write **int arr[5]={1,2,3};** then the remaining element will get initialize to zero.
- The size is optional only when you assign the values at the declaration time. i.e. int **arr[]={**1,2,3,4,5,6,7};

Above instruction will create array of seven integers and assign values.

- **int arr[5] = {0}** will initialize all elements to zero

Two Dimensional Array Cont…

```
int stud[4][2] = { { 1234, 56 }, { 1212, 33 },
                   { 1434, 80 },{ 1312, 78 }} ;
```

OR

```
int stud[4][2] = { 1234, 56, 1212, 33, 1434, 80,
   1312, 78 } ;
```

Two Dimensional Array Cont…

- Initializing a 2-D array it is necessary to mention the second (column) dimension, whereas the first dimension (row) is optional.
- int arr[2][3] = { 12, 34, 23, 45, 56, 45 } ;

  OR

  int arr[ ][3] = { 12, 34, 23, 45, 56, 45 } ;

  (both Initializations are perfectly acceptable)

Two Dimensional Array Cont…

- int arr[2][ ] = { 12, 34, 23, 45, 56, 45 } ;

  OR

- int arr[ ][ ] = { 12, 34, 23, 45, 56, 45 } ;

  (**would never work**.)

Memory Map of a 2-Dimensional Array

(considering size-of each array element = 2 bytes)

s[0][0]	s[0][1]	s[1][0]	s[1][1]	s[2][0]	s[2][1]	s[3][0]	s[3][1]
1234	56	1212	33	1434	80	1312	78
65508	65510	65512	65514	65516	65518	65520	65522

Address calculation for 1-D Array

- Address of ith element in Array:

$$A[i] = A + w * i$$

Where

A = Base Address

w = Size of each element

int A[4];

A[0] = 1000 + 4 * 0 = 1000

A[1] = 1000 + 4 * 1 = 1004

A[2] = 1000 + 4 * 2 = 1008

A[3] = 1000 + 4 * 3 = 1012

Address calculation for 2-D Array

- A[i][j] = A + *w* [N (i) + j]

Where

A = Base Address

w = Size of each element

N = Total no. of columns

int A[2][2];

A[0][0] = 1000 + 4 [2(0) + 0] = 1000 + 0 = 1000

A[0][1] = 1000 + 4 [2(0) + 1] = 1000 + 4 = 1004

A[1][0] = 1000 + 4 [2(1) + 0] = 1000 + 8 = 1008

A[1][1] = 1000 + 4 [2(1) + 1] = 1000 + 12 = 1012

To display the matrix

```
int a[3][2]={{1,2},{3,4},{5,6}};
for(int i=0; i<3; i++)
{
    for(int j=0; j<2; j++)
    {
        printf("%d", a[i][j]);
    }
    printf("\n");
}
```

Out put:

```
A array is:
1 2
3 4
5 6
```

Array Applications

1. Matrix addition

```
// compute c = a + b.
for(int i=0; i<3; i++)
  {
    for(int j=0; j<2; j++)
    {
        c[i][j] = a[i][j] + b[i][j];
    }
  }
```

2. Matrix subtraction

```
for(int i=0; i<3; i++)
 {
   for(int j=0; j<2; j++)
   {
            c[i][j] = a[i][j] - b[i][j];
   }
 }
```

3. Matrix multiplication

```
int A[m][n], B[p][q], C[m][q];

for(int i=0; i<m; i++)
 {
    for(int j=0; j<q; j++)
    {
          sum = 0;
          for(int k=0; k<n; k++)
          {
                    sum+= A[i][k]*B[k][j];
          }
         C[i][j] = sum;
    }
}
```

4. Bubble Sort

17 6 13 12 2		i i+1
6 17 13 12 2		0 - 1
6 13 17 12 2		1 - 2
6 13 12 17 2		2 - 3
6 13 12 2 17		3 - 4
6 13 12 2	17	0 - 1
6 12 13 2	17	1 - 2
6 12 2 13	17	2 - 3
6 12 2	13 17	0 - 1
6 2 12	13 17	1 - 2
2 6	12 13 17	0 - 1

Algorithm: **Bubble Sort**

```
begin BubbleSort(list)

        repeat size-1
                for all elements of list
                        if list[i] > list[i+1]
                                swap(list[i], list[i+1])
                        end if
                end for
        end repeat

return list
end BubbleSort
```

Bubble Sort

```
main( )
{
    int a[ ] = { 17, 6, 13, 12, 2 } ;
    int i, j, t ;
    for ( j = 0 ; j < 4 ; j++ )
    {
      for ( i = 0 ; i <  4  - j; i++ )
      {
        if ( a[ i ] > a[ i + 1 ] )
        {
          t =a[ i ] ; a[ i ] = a[ i + 1 ];
          a[ i + 1 ] = t ;
        }
      }
    }
    for ( i = 0 ; i <= 4 ; i++ )
      printf(“%d”, a[ i ] );
}
```

17 6 13 12 2		i i+1
6 17 13 12 2		0 - 1
6 13 17 12 2		1 - 2
6 13 12 17 2		2 - 3
6 13 12 2 17		3 - 4
6 13 12 2	17	0 - 1
6 12 13 2	17	1 - 2
6 12 2 13	17	2 - 3
6 12 2	13 17	0 - 1
6 2 12	13 17	1 - 2
2 6	12 13 17	0 - 1