11/7/2015

ARRAYS

- An array is a derived data type in 'C' which is constructed from fundamental data type of 'C' Prog. Language.
- An array is a collection of similar types of data elements in a single entity.
- In implementation when we require 'n' no. of values of same data type, then recommended to create an array.
- When we are working with arrays always static memory allocation will happen i.e compile time memory management.
- When we are working with arrays always memory is constructed in continuous memory location that's why possible to access the data randomly.
- When we are working with arrays all values will share same name with unique identification value called 'index').
- Always array index must be required to start with '0' & ends with (size-1).
- When we are working with arrays we required to use array subscript operator i.e [].
- Always array subscript operator require 1 argument of type unsigned integer constant, whose value is always '>0' only.

Syntax `Datatype arr [size];`

Properties of 1D Array

Size → no. of elements, sizeof → no. of bytes

1. `int arr[5];`
   Size → 5
   sizeof(arr) → 10B (5 * 2 = 10B)

2. `int arr[4];`
   Size → 4
   Sizeof(arr) → 4B
   (1 element)

3. int arr[]; error

4. int arr [0]; error

5. int arr [-5]; error

- In declaration of array size must be require to specify or else it gives an error i.e size is unknown.
- In declaration of array size must be unsigned integer constant, whose value is '> 0' only.

6. int arr [5] = { 20, 10, 30, 40, 50 };

   20 → arr [0];
   10 → arr [1];
   30 → arr [2];
   40 → arr [3];
   50 → arr [4];

7. int arr[5] = { 10, 20, 30 }

   arr [0] → 10
   arr [1] → 20
   arr [2] → 30
   arr [3] → 0
   arr [4] → 0

- In initialisation of array, if specified no. of elements are not initialised, the remaining all elements are automatically initialized with zero

8. int arr [2] = { 10, 20, 30, 40, 50 }; error
   In initialization of array we can't initially more than size of array elements, if we are initializing then it gives an error i.e too many initializations.

9. int arr [] = {10, 20, 30, 40, 50}; yes valid

   Size → 5;
   Size of (arr) → 10 B

→ • In initialisation of array specifying the size is optional, in this case how many elements are initialized that many variables are created automatically.

10. int arr [5];

arr [0] = 10;
arr [2] = 30;
arr [4] = 50;

printf ("%d %d", arr[1], arr[8]);

O/P: gr → In declaration of the array by default all elements are having garbage values only bcz, by default it is autotype.

11. Static int arr[5];
arr[0] = 10;
arr[1]
arr[2]
printf ("%d %d", arr[3], arr[4]);

O/P: 0 0

12. int arr[2]
arr[0] = 10;
arr[1] = 20;
arr[2] = 30;
printf ("%d %d %d", arr[0], arr[1], arr[2]);

Valid

In C & CPP there is no any upper boundary checking process occurs, so when we are crossing the limit then depending upon OS, security level anything can happen (segmentation fault).

13. float arr[5.8]; (error)

14. float arr[5];
size → 5
sizeof (arr) → 5X4 = 20B.

→ On DOS based Compiler at compile time we can create maximum of 64KB data i.e 65536B but in previous syntax we require 80,000B

16) long int arr[20000]; Error
20000 X 4 ⇒ 80,00B

17) char arr[4000] Valid

18) int size = 10;
int arr[size], error

19) Const int size = 10
int arr[size]; (error)

20) # define size 10
int arr[size]; (Valid)

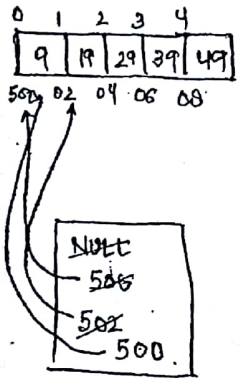
- In declaration of array size can't be variable or constant variable type.
- In declaration of array size can be symbolic constant value because at the time of preprocessing it is replaced with constant value.

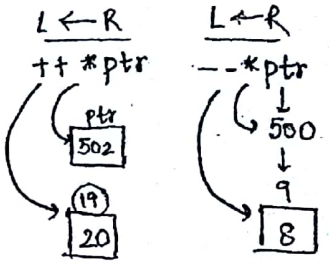
21) int arr [2+3]; Yes, valid
int arr [5];

22) int arr [2>5]; error // int arr [0];

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
int main()
{
  int arr[5] = {9,19,29,39,49}
  int near *ptr = (int near*) NULL ; // will take 2B
                    (equal priority for binary→ )
  ptr = &arr[0];
  ++ *ptr;
  --*ptr;
  printf("%d %d", arr[0], arr[1]);
  getch();
  return EXIT_SUCCESS
}
```

O/P: 8 20





(Pre ++ & * have same priority

* indirection operator & pre-operator both are having equal priority.
* When equal is occured, for unary operator it should be required to evaluate from Right to left only.
* Arithmatic operation of the pointers always data type dependent only.

```
#include <stdio.h>
#include <conio.h>
int main(void)
{
  int arr[5] = {5,15,25,35,45};
  int *ptr = (int *) NULL; // ptr will take 4B (32 bit compiler)
  ptr = &arr[1];
  --ptr;
  --*ptr;
```

Programs:- Array

1) WAP to enter roll number of 10 students.

```
void Main()
  {
   int roll[10], i;
   Printf("\n Enter roll number");
   for( i=0; i<10; i++)
     {
      Scanf("%d", &roll[i]);
     }
   Printf("\n you entered: \n");
   for (i=0; i<10; i++)
     {
      Printf("%d\n", roll[i]);
     }
   getch();
  }
```

2) Insert 10 number in an array and Print the sum of these number.

```
void Main()
  {
    int num[10], i, sum=0;
    printf("\n Enter the numbers");
    for( i=0; i<10; i++)
      {
       Scanf("%d", &num[i]);
      }
    for( i=0; i<10; i++)
      {
        sum= sum + num[i];
      }
```

```
}
Printf("%d\n", sum);
}
}
```

3) WAP enter 10 numbers and add the values with a particular number entered by the user.

```
void main()
{
int num[10], i, dg;
Printf("\n Enter the numbers");
for(i=0; i<10; i++)
{
Sf("%d", &num[i]);
}
for(i=0; i<10; i++)
{
num[i] = num[i] + dg;
}
Printf("%d\n", num[i]);
}
```

need to add one more for loop

4) Enter 10 numbers Print the average.

```
void main()
{
int n[10], i, sum=0;
float avg;
Pf("\n Enter the values");
for(i=0; i<10; i++)
{
Sf("%d", &n[i]);
}
for(i=0; i<10; i++)
{
```

```
    sum = sum + n[i];
  }
avg = sum/10;
Pf("\n Average = %f", avg);
}
```

5) Enter the numbers and print which is even or odd.

```
void Main()
{
int n[10], i;
Pf("\n enter the values: \n");
for(i=0; i<10; i++)
  {
  Sf("%d", &n[i]);
  }
for(i=0; i<10; i++)
  {
  If (n[i] % 2 == 0)
  Pf ("%d is even", n[i]);
  else
    Pf ("%d is odd", n[i]);
}
```

Application of Array :-

1) Insert
2) Delete
3) Update
4) Search
5) Sort.

Insert:-

a[5]

0	1	2	3	4	5
10	20	30	40	50	

po = 3

n1 = 60

after insertion a[6].

```
void Main()
{
    int a[50], n, po, i, n1;
    pf("Enter the range");
    sf("%d", &n);
    pf("Enter the elements");
    for(i=0; i<n; i++)
    {
        sf("%d", &a[i]);
    }
    pf("Enter the position to insert");
    sf("%d", &po);
    pf("Enter the value to insert");
    sf("%d", &n1);
    for(i=n-1; i>=po-1; i--)
    {
        a[i+1] = a[i];
    }
    a[po-1] = n1;
    n++;
    for(i=0; i<n; i++)
    pf("%d", a[i]);
}
```

0	1	2	3	4	5
10	20	60	30	40	50

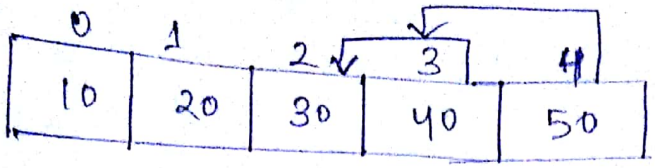
n = 5 6

po = 3

n1 = 60

i = 4 3 2

delete:-



PO = 3

n = 5 4
i = 3

```
void main()
{
  int a[50], n, po, i;
  pf("enter the range");
  sf("%d", &n);
  pf("enter the elements");
  for(i=0; i<n; i++)
  sf("%d", &a[i]);
  pf("enter the position to delete");
  sf("%d", &po);
  for(i=po-1; i<n-1; i++)
  {
      a[i] = a[i+1];
  }
  n--;
  for(i=0; i<n; i++)
  pf("%d", a[i]);
}
```

~~0 1 2 3 4 5 6 7 8 9 10~~