UNIT II

Arrays in C

When we work with large number of data values we need that many number of different variables. As the number of variables increases, the complexity of the program also increases and so the programmers get confused with the variable names. There may be situations where we need to work with large number of similar data values. To make this work more easy, C programming language provides a concept called "Array".

An array is a special type of variable used to store multiple values of same data type at a time.

An array can also be defined as follows...

An array is a collection of similar data items stored in continuous memory locations with single name which are accessed by the reference indexed number is called array.

Declaration of an Array

In c programming language, when we want to create an array we must know the datatype of values to be stored in that array and also the number of values to be stored in that array.

- We use the following general syntax to create an array...

datatypearrayName [size] ;

- Syntax for creating an array with size and initial values

datatypearrayName [size] = {value1, value2, ...} ;

- Syntax for creating an array without size and with initial values

datatypearrayName [ ] = {value1, value2, ...} ;

In the above syntax, the datatype specifies the type of values we store in that array and size specifies the maximum number of values that can be stored in that array.

Example

int a [3] ;

P.VAMSHEEDHAR REDDY
(Asst.Prof,CSE DEPT)

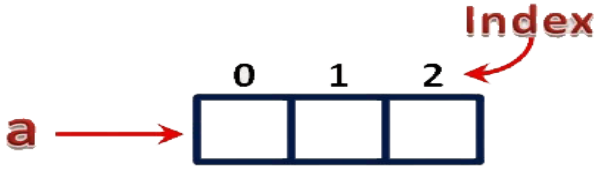
(If any data needed to add into this material please write to : pvamsheedharreddy@gmail.com)

UNIT II

Here, the compiler allocates 6 bytes of continuous memory locations with single name 'a' and tells the compiler to store three different integer values (each in 2 bytes of memory) into that 6 bytes of memory.
For the above declaration the memory is organized as follows...

- In the above memory allocation, all the three memory locations has a common name 'a'.
- So the accession of individual memory location is not possible directly.
- Hence compiler not only allocates the memory but also assigns a numerical reference value to every individual memory location of an array.
- This reference number is called as "Index" or "subscript" or "indices".

Index values for the above example is as follows...



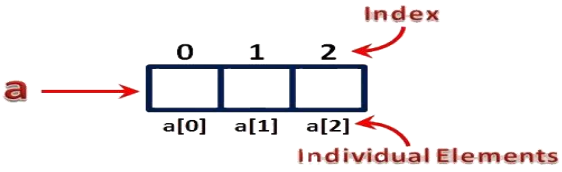
Accessing Individual Elements of an Array:

The individual elements of an array are identified using the combination of 'arrayName' and 'indexValue'.

We use the following general syntax to access individual elements of an array...

arrayName [indexValue] ;

For the above example the individual elements can be denoted as follows...



For example, if we want to assign a value to the second memory location of above array 'a', we use the following statement...

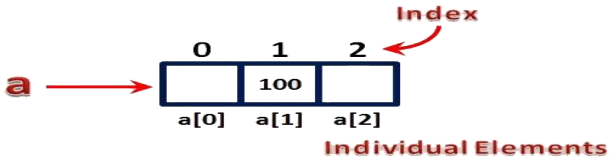
a [1] = 100 ;

P.VAMSHEEDHAR REDDY
(Asst.Prof,CSE DEPT)

(If any data needed to add into this material please write to : pvamsheedharreddy@gmail.com)

UNIT II

The result of above assignment statement is as follows...



Types of Arrays in C

In c programming language, arrays are classified into two types. They are as follows...

1. **Single Dimensional Array / One Dimensional Array**
2. **Multi Dimensional Array**

Single Dimensional Array

- In c programming language, single dimensional arrays are used to store list of values of same datatype.
- In other words, single dimensional arrays are used to store a row of values.
- In single dimensional array, data is stored in linear form. Single dimensional arrays are also called as one-dimensional arrays, Linear Arrays or simply 1-D Arrays.

Declaration of Single Dimensional Array

We use the following general syntax for declaring a single dimensional array...

datatypearrayName [size] ;

Example

introllNumbers [60] ;

The above declaration of single dimensional array reserves 60 continuous memory locations of 2 bytes each with the name rollNumbers and tells the compiler to allow only integer values into those memory locations.

Initialization of Single Dimensional Array

We use the following general syntax for declaring and initializing a single dimensional array with size and initial values.

datatypearrayName [size] = {value1, value2, ...} ;

UNIT II

Example

int marks [6] = { 89, 90, 76, 78, 98, 86 } ;

The above declaration of single dimensional array reserves 6 continuous memory locations of 2 bytes each with the name marks and initializes with value 89 in first memory location, 90 in second memory location, 76 in third memory location, 78 in fourth memory location, 98 in fifth memory location and 86 in sixth memory location.

We can also use the following general syntax to intialize a single dimensional array without specifying size and with initial values...

datatypearrayName [ ] = {value1, value2, ...} ;

The array must be initialized if it is created without specifying any size. In this case, the size of the array is decided based on the number of values initialized.

Example

int marks [] = { 89, 90, 76, 78, 98, 86 } ;

charstudentName [] = "btechsmartclass" ;

In the above example declaration, size of the array 'marks' is 6 and the size of the array 'studentName' is 16.

This is because in case of character array, compiler stores one exttra character called **\0 (NULL)** at the end.

Accessing Elements of Single Dimensional Array

- In c programming language, to access the elements of single dimensional array we use array name followed by index value of the element that to be accessed.
- Here the index value must be enclosed in square braces. Index value of an element in an array is the reference number given to each element at the time of memory allocation.
- The index value of single dimensional array **starts with zero (0)** for first element and incremented by one for each element.
- The index value in an array is also called as **subscript or indices**.

We use the following general syntax to access individual elements of single dimensional array...

arrayName [indexValue]

UNIT II

Example

marks [2] = 99 ;

In the above statement, the third element of 'marks' array is assinged with value '99'.

Example Program for Single Dimensional Array:

Programm:

```
#include<stdio.h>

#include<conio.h>

int main()

{

inti,number[5];

clrscr();

printf(“Enter 5 number \n”);

for(i=0;i<5;i++)

scanf(“%d”,&numbers[i]);

printf(“Array elements are \n”);

for(i=0;i<5;i++)

printf(“%d\n”,numbers[i]);

getch();

return 0;

}
```

UNIT II

Multi Dimensional Array

- An array of arrays is called as multi dimensional array.
- In simple words, an array created with more than one dimension (size) is called as multi dimensional array.
- Multi dimensional array can be of two dimensional array or three dimensional array or four dimensional array or more...

- Most popular and commonly used multi dimensional array is two dimensional array. The 2-D arrays are used to store data in the form of table. We also use 2-D arrays to create mathematical matrices.

Declaration of Two Dimensional Array

We use the following general syntax for declaring a two dimensional array...

datatypearrayName [rowSize] [columnSize] ;

Example

intmatrix_A [2][3] ;

The above declaration of two dimensional array reserves 6 continuous memory locations of 2 bytes each in the form of 2 rows and 3 columns.

Initialization of Two Dimensional Array

We use the following general syntax for declaring and initializing a two dimensional array with specific number of rows and coloumns with initial values.

datatypearrayName [rows][colmns] = {{r1c1value, r1c2value, ...},{r2c1, r2c2,...}...} ;

Example

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

The above declaration of two dimensional array reserves 6 continuous memory locations of 2 bytes each in the form of 2 rows and 3 coloumns. And the first row is initialized with values 1, 2 & 3 and second row is initialized with values 4, 5 & 6.

We can also initialize as follows...

```
intmatrix_A [2][3] = {
                        {1, 2, 3},
                        {4, 5, 6}
```

P.VAMSHEEDHAR REDDY
(Asst.Prof,CSE DEPT)

(If any data needed to add into this material please write to : pvamsheedharreddy@gmail.com)

UNIT II

} ;

Accessing Individual Elements of Two Dimensional Array

In c programming language, to access elements of a two dimensional array we use array name followed by row index value and column index value of the element that to be accessed. Here the row and column index values must be enclosed in separate square braces. In case of two dimensional array the compiler assigns separate index values for rows and columns.

We use the following general syntax to access the individual elements of a two dimensional array...

arrayName [rowIndex] [columnIndex]

Example

matrix_A [0][1] = 10 ;

In the above statement, the element with row index 0 and column index 1 of matrix_A array is assinged with value 10.

Example Program for TWO Dimensional Array:

```
#include <stdio.h>
#include<conio.h>

int main()
{
int m, n, c, d, first[10][10], second[10][10], sum[10][10];

printf("Enter the number of rows and columns of matrix\n");
scanf("%d%d", &m, &n);
printf("Enter the elements of first matrix\n");

for (c = 0; c < m; c++)
for (d = 0; d < n; d++)
scanf("%d", &first[c][d]);

printf("Enter the elements of second matrix\n");

for (c = 0; c < m; c++)
for (d = 0 ; d < n; d++)
scanf("%d", &second[c][d]);

printf("Sum of entered matrices:-\n");
```

UNIT II

```
for (c = 0; c < m; c++) {
for (d = 0 ; d < n; d++) {
sum[c][d] = first[c][d] + second[c][d];
printf("%d\t", sum[c][d]);
     }
printf("\n");
  }

return 0;
}
```

Pointers in C

In c programming language, we use normal variables to store user data values. When we declare a variable, the compiler allocates required memory with specified name.

In c programming language, every variable has name, datatype, value, storage class, and address. We use a special type of variable called pointer to store the address of another variable with same datatype.

Pointer is defined as follows...

Definition:

Pointer is a special type of variable used to store the memory location address of a variable.

In c programming language, we can create pointer variables of any datatype. Every pointer stores the address of variable with same datatype only. That means, integer pointer is used store the address of integer variable only.

Accessing the Address of Variables

In c programming language, we use the **reference operator "&"** to access the address of variable.

For example, to access the address of a variable "marks" we use "&marks". We use the following printf statement to display memory location address of variable "marks"...

printf("Address : %u", &marks) ;

In the above example statement %u is used to display address of marks variable. Address of any memory location is unsigned integer value.

Declaring Pointers (Creating Pointers)

UNIT II

In c programming language, declaration of pointer variable is similar to the creation of normal variable but the name is prefixed with * symbol. We use the following syntax to declare a pointer variable...

datatype *pointerName ;

A variable declaration prefixed with * symbol becomes a pointer variable.

Example

int *ptr ;
In the above example declaration, the variable "ptr" is a pointer variable that can be used to store any integer variable address.

Assigning Address to Pointer

To assign address to a pointer variable we use assignment operator with the following syntax...

pointerVariableName = & variableName ;

For example, consider the following variables declaration...

int a, *ptr ;
In the above declaration, variable "a" is a normal integer variable and variable "ptr" is an integer pointer variable. If we want to assign the address of variable "a" to pointer variable "ptr" we use the following statement...

ptr = &a ;
In the above statement, the address of variable "a" is assigned to pointer variable "prt". Here we say that pointer variable ptr is pointing to variable a.

Accessing Variable Value Using Pointer

Pointer variables are used to store the address of other variables. We can use this address to access the value of the variable through its pointer. We use the symbol **de-reference operator"*"** infront of pointer variable name to access the value of variable to which the pointer is pointing.

We use the following general syntax...

***pointerVariableName**

Example

#include<stdio.h>

UNIT II

```
#include<conio.h>

void main()
{
  int a = 10, *ptr ;
  clrscr();
  ptr = &a ;

  printf("Address of variable a = %u\n", ptr) ;
  printf("Value of variable a = %d\n", *ptr) ;
  printf("Address of variable ptr = %u\n", &ptr) ;
}
```

Output

```
Address of variable a = 65524
Value of variable a = 10
Address of variable ptr = 65526
```

In the above example program, variable a is a normal variable and variable ptr is a pointer variable. Address of variable a is stored in pointer variable ptr. Here ptr is used to access the address of variable a and *ptr is used to access the value of variable a.

Memory Allocation of Pointer Variables

Every pointer variable is used to store the address of another variable. In computer memory address of any memory location is an unsigned integer value. In c programming language, unsigned integer requires 2 bytes of memory. So, irrespective of pointer datatype every pointer variable is allocated with 2 bytes of memory

UNIT II

Pointers Arithmatic Operations in C

Pointer variables are used to store address of variables. Address of any variable is an unsigned integer value i.e., it is a numerical value. So we can perform arithematic operations on pointer values. But when we perform arithematic operations on pointer variable, result depends on the amount of memory required by the variable to which the pointer is pointing.

In c programming language, we can perform the following arithematic operations on pointers...

1. Addition
2. Subtraction
3. Increment
4. Decrement
5. Comparison

Addition Operation on Pointer

In c programming language, the addition operation on pointer variables is calculated using the following formula...

AddressAtPointer + (NumberToBeAdd * BytesOfMemoryRequiredByDatatype)

Example

```
#include<stdio.h>
#include<conio.h>

void main()
{
  int a, *intPtr ;
  float b, *floatPtr ;
  double c, *doublePtr ;
  clrscr() ;
  intPtr = &a ;  // Asume address of a is 1000
  floatPtr = &b ;  // Asume address of b is 2000
  doublePtr = &c ;  // Asume address of c is 3000

  intPtr = intPtr + 3 ;  // intPtr = 1000 + ( 3 * 2 )
  floatPtr = floatPtr + 2 ;  // floatPtr = 2000 + ( 2 * 4 )
  doublePtr = doublePtr + 5 ;  // doublePtr = 3000 + ( 5 * 6 )

  printf("intPtr value : %u\n", intPtr) ;
  printf("floatPtr value : %u\n", floatPtr) ;
  printf("doublePtr value : %u", doublePtr) ;
```

UNIT II

```
  getch() ;
}
intPtr value : 1006
floatPtr value : 2008
doublePtr value : 3030
```

Subtraction Operation on Pointer

In c programming language, the subtraction operation on pointer variables is calculated using the following formula...

AddressAtPointer - (NumberToBeAdd * BytesOfMemoryRequiredByDatatype)

Example

```
#include<stdio.h>
#include<conio.h>

void main()
{
  int a, *intPtr ;
  float b, *floatPtr ;
  double c, *doublePtr ;
  clrscr() ;
  intPtr = &a ;  // Asume address of a is 1000
  floatPtr = &b ;  // Asume address of b is 2000
  doublePtr = &c ;  // Asume address of c is 3000

  intPtr = intPtr - 3 ;  // intPtr = 1000 - ( 3 * 2 )
  floatPtr = floatPtr - 2 ;  // floatPtr = 2000 - ( 2 * 4 )
  doublePtr = doublePtr - 5 ;  // doublePtr = 3000 - ( 5 * 6 )

  printf("intPtr value : %u\n", intPtr) ;
  printf("floatPtr value : %u\n", floatPtr) ;
  printf("doublePtr value : %u", doublePtr) ;

  getch() ;
}
intPtr value : 994
floatPtr value : 1992
doublePtr value : 2970
```

UNIT II

Increment & Decrement Operation on Pointer

The increment operation on pointer variable is calculated as follows...

AddressAtPointer + NumberOfBytesRequiresByDatatype

Example

```
#include<stdio.h>
#include<conio.h>

void main()
{
  int a, *intPtr ;
  float b, *floatPtr ;
  double c, *doublePtr ;
  clrscr() ;
  intPtr = &a ;  // Asume address of a is 1000
  floatPtr = &b ;  // Asume address of b is 2000
  doublePtr = &c ;  // Asume address of c is 3000

  intPtr++ ;  // intPtr = 1000 + 2
  floatPtr++ ;  // floatPtr = 2000 + 4
  doublePtr++ ;  // doublePtr = 3000 + 6

  printf("intPtr value : %u\n", intPtr) ;
  printf("floatPtr value : %u\n", floatPtr) ;
  printf("doublePtr value : %u", doublePtr) ;

  getch() ;
}
intPtr value : 1002
floatPtr value : 2004
doublePtr value : 3006
```

The decrement operation on pointer variable is calculated as follows...

AddressAtPointer - NumberOfBytesRequiresByDatatype

Example

```
#include<stdio.h>
#include<conio.h>
```

UNIT II

```
void main()
{
  int a, *intPtr ;
  float b, *floatPtr ;
  double c, *doublePtr ;
  clrscr() ;
  intPtr = &a ;  // Asume address of a is 1000
  floatPtr = &b ;  // Asume address of b is 2000
  doublePtr = &c ;  // Asume address of c is 3000

  intPtr-- ;  // intPtr = 1000 - 2
  floatPtr-- ;  // floatPtr = 2000 - 4
  doublePtr-- ;  // doublePtr = 3000 - 6

  printf("intPtr value : %u\n", intPtr) ;
  printf("floatPtr value : %u\n", floatPtr) ;
  printf("doublePtr value : %u", doublePtr) ;

  getch() ;
}
```

```
intPtr value : 998
floatPtr value : 1996
doublePtr value : 2994
```

Comparison of Pointers

The comparison operation is perform between the pointers of same datatype only. In c programming language, we can use all comparison operators (relational operators) with pointers.

We can't perform multiplication and division operations on pointers.

UNIT II

Pointers to Pointers in C

In c programming language, we have pointers to store the address of variables of any datatype. A pointer variable can store the address of normal variable.

Definition:
C programming language also provides pointer variable to store the address of another pointer variable. This type of pointer variable is called as pointer to pointer variable.

Sometimes we also call it as double pointer. We use the following syntax for creating pointer to pointer…

Syntax:

datatype **pointerName ;

Example

int **ptr ;

Here, ptr is an integer pointer variable that stores the address of another integer pointer variable but does not stores the normal integer variable address.

Points to be Remembered

1. To store the address of normal variable we use single pointer variable
2. To store the address of single pointer variable we use double pointer variable
3. To store the address of double pointer variable we use triple pointer variable
4. Similarly the same for remaining pointer variables also…

Example Program

```
#include<stdio.h>
#include<conio.h>

void main()
{
  int a ;
  int *ptr1 ;
  int **ptr2 ;
  int ***ptr3 ;

  clrscr() ;

  ptr1 = &x ;
```

```
  ptr2 = &ptr1 ;
  ptr3 = &ptr2 ;

  printf("Address of 'x' = %u\n", ptr1) ;
  printf("Address of 'ptr1' = %u\n", ptr2) ;
  printf("Address of 'ptr2' = %u\n", ptr3) ;

  getch() ;
}
```

Pointers to void in C

In c programming language, pointer to void is the concept of defining a pointer variable that is independent of datatype.

In C programming language, void pointer is a pointer variable used to store the address of variable of any datatype.

That means single void pointer can be used to store address of integer variable, float variable, character variable, double variable or any structure variable.

We use the keyword "void" to create void pointer. We use the following syntax for creating pointer to void…

void *pointerName ;

Example

void *ptr ;
Here, "ptr" is a void pointer variable which is used to store the address of any datatype variable.

Points to be Remembered

void pointer stores the address of any datatype variable.
Example Program

```
#include<stdio.h>
#include<conio.h>

void main()
{
  int a ;
  float b ;
  char c ;
```

UNIT II

```
  void *ptr ;

  clrscr() ;

  ptr = &a ;
  printf("Address of integer variable 'a' = %u\n", ptr) ;

  ptr = &b ;
  printf("Address of float variable 'b' = %u\n", ptr) ;

  ptr = &c ;
  printf("Address of character variable 'c' = %u\n", ptr) ;

  getch() ;
}
```

Pointers to Arrays in C

In c programming language, when we declare an array the compiler allocates required amount of memory and also creates constant pointer with array name and stores **the base address** of that pointer in it.

The address of first element of an array is called as base address of that array.

The array name itself acts as pointer to the first element of that array.

Consider the following example of array declaration...

int marks[6] ;

For the above declaration, the compiler allocates 12 bytes of memory and the address of first memory location (i.e., marks[0]) is stored in a constant pointer called marks.

That means in the above example, marks is a pointer to marks[0].

Example Program

```
#include<stdio.h>
#include<conio.h>

void main()
```

UNIT II

```
{
  int marks[6] = {89, 45, 58, 72, 90, 93} ;
  int *ptr ;

  clrscr() ;
  ptr = marks ;
  printf("Base Address of 'marks' array = %u\n", ptr) ;
getch() ;
}
```

Points to be Remembered

- An array name is a constant pointer.
- We can use the array name to access the address and value of all the elements of that array.
- Since array name is a constant pointer we can't modify its value.
-

Consider the following example statements...
ptr = marks + 2 ;

Here, the pointer variable "ptr" is assigned with address of "marks[2]" element.

printf("Address of 'marks[4]' = %u", marks+4) ;

The above printf statement displays the address of element "marks[4]".

printf("Value of 'marks[0]' = %d", *marks) ;
printf("Value of 'marks[3]' = %d", *(marks+3)) ;

In the above two statements, first printf statement prints the value 89 (i.e., value of marks[0]) and the second printf statement prints the value 72 (i.e., value of marks[3]).

marks++ ;
The above statement generates compilation error because the array name acts as a constant pointer. So we can't change its value.

In the above example program, the array name marks can be used as follows...

marks is same as &marks[0]
marks + 1 is same as &marks[1]
marks + 2 is same as &marks[2]
marks + 3 is same as &marks[3]
marks + 4 is same as &marks[4]
marks + 5 is same as &marks[5]
*marks is same as marks[0]

*(marks + 1) is same as marks[1]
*(marks + 2) is same as marks[2]
*(marks + 3) is same as marks[3]
*(marks + 4) is same as marks[4]
*(marks + 5) is same as marks[5]

Pointers to Multi Dimensional Array

In case of multi dimensional array also the array name acts as a constant pointer to the base address of that array.

For example, we declare an array as follows...

int marks[3][3] ;

In the above example declaration, the array name marks acts as constant pointer to the base address (address of marks[0][0]) of that array.
In the above example of two dimensional array, the element marks[1][2] is accessed as *(*(marks + 1) + 2).

Pointers For Functions in C

In c programming language, there are two ways to pass parameters to functions. They are as follows...

1. Call by Value
2. Call By Reference
3.

We use pointer variables as formal parameters in call by reference parameter passing method. In case of call by reference parameter passing method, the address of actual parameters is passed as arguments from the calling function to the called function. To recieve this address, we use pointer variables as formal parameters.

Consider the following program for swapping two variable values...

Example - Swapping of two variable values using Call by Reference

```
#include<stdio.h>
#include<conio.h>

void swap(int *, int *) ;
```

UNIT II

```
void main()
{
  int a = 10, b = 20 ;

  clrscr() ;

  printf("Before swap : a = %d and b = %d\n", a, b) ;

  swap(&a, &b) ;

  printf("After swap : a = %d and b = %d\n", a, b) ;

  getch() ;
}
void swap(int *x, int *y)
{
  int temp ;
  temp = *x ;
  *x = *y ;
  *y = temp ;
}
```

Output

```
Before swap : a = 10 and b = 20
After swap : a = 20 and b = 10
```

In the above example program, we are passing the addresses of variables a and b and these are recieved by the pointer variables x and y. In the called function swap we use the pointer variables x and y to swap the values of variables a and b.