26/03/2017

prof. chaithra H.t

Module 4 :-

Arrays (part 1)

* Introduction
* Declaratn & initializatn of 1-D array
* Reading & printing of 1-D array
* Bubble sort
* Linear & Binary search
* Reading & printing of 2-D array
* programs on matrix operations: Addition, subtraction, multiplication, transpose

Introduction :-

```
int marks = 20;
```

In the above declaration, variables marks stores only one student marks.

Suppose if we want to store marks of 100 students, then we cant declare 100 variables. Instead of that we can use array.

we use arrays to store set of data items of same type.

```
int marks[90];
```

In this, 90 fields will be allocated to store marks.

* Array :- Array is an ordered set of similar data items.

eg:- int marks[5];

Here, 5 fields will be allocated.

0	1	2	3	4
50	70	80	100	90
marks[0]	marks[1]	marks[2]	marks[3]	marks[4]

marks[0] = 50 → marks of 1st student
marks[4] = 90 → marks of 5th student

i.e., value can be accessed by using array index.

array index starts with 0 (zero).

In this example marks of 5 students are accessed by specifying marks[0] till marks[n-1] = marks[4].

Types of arrays

1) single dimensional array (1-D array)
2) Multi dimensional array (2-D, 3-D)

1) Single-dimensional array (1-D array)

It is a linear list consists of similar data items.

In memory data items are stored contiguos one after the other.

Array size depends on the data type we store.

```
int a[5];
```

In this, array is 10 bytes.

i.e., Array size = n * sizeof(datatype)
= 5 * sizeof(int);
= 5 * 2
= 10 bytes

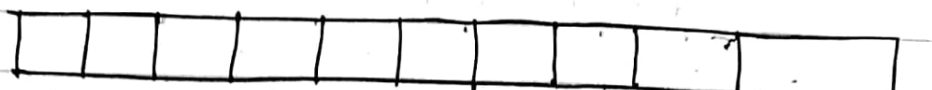
** Declaration of 1-D array

Syntax :- data_type array_name[size];

Eg:- int a[10]; (size: positive integer)

10 spaces will be reserved for a locations.

```
int a[10];
```



** Initialization of 1-D array

* Assigning values to an array.

Syntax :- data_type array_name[size] = {v1, ---, vn};

(v1, ---, vn: variable values.)

* data types can be int, float, char etc.,
* array_name is name of array
* size or expression should be always gives positive integer.

Different ways of initializing arrays

* * Compile time initialization :-

1) Initializing all specified memory locations

Arrays can be initialized at the time of declaration when their initial values are known in advance.

eg:- `int a[5] = { 10, 15, 20, 25, 30 };`

During compilation, 5 contiguos memory locations are reserved by the compiler for variable a & all these locations are initialized as shown below :-

a[0]	a[1]	a[2]	a[3]	a[4]
10	15	20	25	30

For characters,

`int b[5] = { 'H', 'E', 'L', 'L', 'O' };`

b[0]	b[1]	b[2]	b[3]	b[4]
H	E	L	L	O

2) partial array initialization.

Here, number of values initialized is less than the size of array.

Elements are initialized in the order from 0th location. The remaining locations will be initialized to zero automatically.

Eg:- int a[5] = {10, 20};

5 memory locations will be allocated. & compiler initializes first 2 location with 10 & 20. The remaining memory locations are automatically initialized to 0's.

a[0]	a[1]	a[2]	a[3]	a[4]
10	20	0	0	0

3) Initialization without size

Consider,

char b[] = {'H', 'E', 'L', 'L', 'O'};

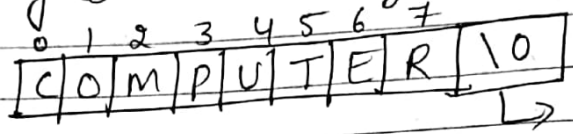
In this example, the array size is not specified. But array size will be set to total number of initial values specified. i.e., array size is automatically set to 5.

b[0]	b[1]	b[2]	b[3]	b[4]
H	E	L	L	O

4) Array initialization with a string

consider, char b[] = "COMPUTER";

Array b is initialized as below,



String "COMPUTER" contains 8 characters, but string always ends with null characters. so array size is 9 bytes (string length + 1).

** Run time initialization (Reading array)

* Reading & printing one-D array

consider, int a[5];

* 5 memory locations are allocated.
* Each elements can be accessed by specifying index.

* using a[0] through a[4], we can access 5 data items.

* The array can be read by using scanf() as follows:

```
scanf("%d", &a[0]);
:
:
scanf("%d", &a[n-1]);
```

In general, scanf("%d", &a[i]) where i = 0, 1, ----, n-1.
We can read n data items from keyboard as follows.

```
for(i=0; i<n; i++)
{
    scanf("%d", &a[i]);
}
```

Similarly, the n data items can be display using printf() as follows:

```
for(i=0; i<n; i++)
{
    printf("%d", a[i]);
}
```

CHAITHRA. H.E.
B.E, M. Tech

WAP to read n elements & display the same using array.

```
void main()
{
    int n, a[10], i;
    printf("Enter no. of elements \n");
    scanf("%d", &n);
    printf("Enter n elements \n");
    for(i=0; i<n; i++)
    scanf("%d", &a[i]);
```

```
printf("The N elements are \n");
for(i=0; i<n; i++)
printf("%d\t", a[i]);
}
```

WAP to read n elements, find sum & display result using array.

```
void main()
{
  int n, i, a[10], sum=0;
  printf("Enter number of elements \n");
  scanf("%d", &n);
  printf("Enter N elements \n");
  for(i=0; i<n; i++)
  scanf("%d", &a[i]);
  for(i=0; i<n; i++)
  sum=sum+a[i];
  printf("sum=%d", sum);
  getch();
}
```

WAP to find sum of positive numbers, negative number using array.

```
void main()
{
  int n, i, a[10], psum=0, nsum=0;
  clrscr();
```

```
printf("Enter n\n");
scanf("%d", &n);
printf("Enter elements\n");
for(i=0; i<n; i++)
{
   scanf("%d", &a[i]);
}
for(i=0; i<n; i++)
{
   if(a[i]>=0)
      psum = psum+a[i];
   else
      nsum = nsum+a[i];
}
printf("Sum of +ve nos = %d\n", psum);
printf("Sum of -ve nos = %d\n", nsum);
getch();
}
```

CHAITHRA. H.E.
B.E, M. Tech

WAP to input n integer nos & conduct linear search for a given array elements.

```
void main()
{
   int n, i, a[100], key, found=0;
   clrscr();
   printf("Enter number of elements\n");
   scanf("%d", &n);
```

```
printf("Enter elements \n");
for(i=0; i<n; i++)
scanf("%d", &a[i]);
printf("Enter key element to search \n");
scanf("%d", &key);
for(i=0; i<n; i++)
{
  if (a[i]==key)
  {
    found=1;
    break;
  }
}
if (found==1)
printf("key found at position=%d", i+1);
else
printf("not found");
getch();
}
```

CHAITHRA. H.E.
B.E, M. Tech

* Lab program 4(Bubble sort)
* Lab program 5(Binary search)

26/03/2017

Prof. chaithra H.E

Prakash

DATE: PAGE:

WAP to find largest of N elements using array.

```
void main()
{
  int a[10], n, i, big;
  printf("Enter n\n");
  scanf("%d", &n);
  printf("Enter elements\n");
  for(i=0; i<n; i++)
  scanf("%d", &a[i]); → big = a[0];
  for(i=1; i<n; i++)
  {
     if (a[i] > big)
     {
        big = a[i];
     }
  }
  printf("Largest = %d\n", big);
}
```

CHAITHRA. H.E.
B.E, M. Tech

DATE: PAGE:

Two-dimensional array (2D arrays)

It is an array where data items will be stored in the form of rows & columns in a table fashion.

*Declaration :-

syntax :- data-type array-name [exp1][exp2];

where data-type can be int, float, char etc,
* array-name is the name of array.
* exp1 & exp2 are constant expression.

* exp1 specifies rows to be accessed.
* exp2 specifies columns to be accessed.

int a[3][4];

In this, array has 3 rows & 4 columns. (3*4=12) memory locations are allocated.

Pictorial representation of array

	col-0	col-1	col-2	col-3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
ROW 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Prakash
DATE: PAGE:

Initializatn of 2-D array

Syntax:-

data-type array-name[exp1][exp2] = {

{a1, ---, an},
.....
.....

{z1, ---, zn}

};

Here, a1 to an are the values assigned to 1st row & so on.

1) Initializing all specified memory locations

Consider, int a[2][2] = {

{1, 2},

{3, 4}

};

Array a has 2 rows & 2 columns.

pictorial representation of this 2-D array is shown below:-

P.T.O

Columns →

rows ↓	0	1
0	1	2
1	3	4

2) partial array initialization

Here, number of values initialized are less than the size of array.

The remaining locations are initialized to zero automatically.

```
int a[2][3] = {
                { {1, 2},
                  {3, 4}
                };
```

rows ↓	0	1	2
0	1	2	0
1	3	4	0

Reading & printing of 2-D array

To access each item row wise in 2D-array, the row index i should be in outer loop & column index j should be in inner loop.

To read 2-D array of size m×n :-

```
for(i=0; i<m; i++)  /* m - rows */
{
    for(j=0; j<n; j++)  /* n - columns */
    {
        scanf("%d", &a[i][j]);
    }
}
```

↳ /* read m×n matrix */

To print 2-D array of size m×n :-

```
for(i=0; i<m; i++)
{   for(j=0; j<n; j++)
    {   printf("%d", a[i][j]);
    }   printf("\n");
}
```

DATE: PAGE:

* wap to read mxn matrix & display the same.

```
void main()
{
    int m, n, a[10][10], i, j;
    printf("Enter size of matrix\n");
    scanf("%d%d", &m, &n);
    printf("Enter elements of matrix A\n");
    for(i=0; i<m; i++)
    {
        for(j=0; j<n; j++)
        {
            scanf("%d", &a[i][j]);
        }
    }
    printf("Elements of matrix A\n");
    for(i=0; i<m; i++)
    {
        for(j=0; j<n; j++)
        {
            printf("%d\t", a[i][j]);
        }
        printf("\n");
    }
}
```

Prakash

DATE: PAGE

* WAp to add 2 matrices & store result in 3rd matrix.

```
void main()
{
  int m,n,i,j,a[10][10],b[10][10],
  c[10][10];
  printf("Enter size of matrix\n");
  scanf("%d%d",&m,&n);
  printf("Enter elements of matrix A\n");
  for(i=0;i<m;i++)
  {
    for(j=0;j<n;j++)
    {
      scanf("%d",&a[i][j]);
    }
  }
  for(i=0;i<m;i++)
  {
    for(j=0;j<n;j++)
    {
      scanf("%d",&b[i][j]);
    }
  }
  for(i=0;i<m;i++)
  {
    for(j=0;j<n;j++)
    {
      c[i][j]=a[i][j]+b[i][j];
    }
  }
```

```
printf("Resultant matrix\n");
for(i=0;i<m;i++)
{
  for(j=0;j<n;j++)
  {
    printf("%3d", c[i][j]);
  }
  printf("\n");
}
getch();
}
```

CHAITHRA. H.E.
B.E, M. Tech

** * WAP to find transpose of matrix.

```
void main()
{
  int i, j, m, n, a[10][10], t[10][10];
  printf("Enter size of matrix\n");
  scanf("%d%d", &m, &n);
  printf("Enter elements\n");
  for(i=0; i<m; i++)
  {
    for(j=0; j<n; j++)
    {
      scanf("%d", &a[i][j]);
    }
  }
```

```
for(i=0; i<m; i++)
{
 for(j=0; j<n; j++)
  {
    t[i][j] = a[i][j];
  }
}

printf("Transpose matrix\n");
 for(i=0; i<n; i++)
 {
  for(j=0; j<m; j++)
  {
    printf("%d\t", t[j][i]);
  }
  printf("\n");
 }
 getch();
}
```

Note:- You can find transpose of matrix without using t(temporary matrix). P.T.O

DATE PAGE

* sum of elements of matrix pgm.
* lab programs (matrix multiplication
* Trace of matrix.

* WAP to find sum of elements of matrix.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int a[10][10], i, j, m, n, sum = 0;
    clrscr();
    printf("Enter row & column size\n");
    scanf("%d%d", &m, &n);
    printf("Enter matrix elements\n");
    for(i=0; i<m; i++)
    {
    for(j=0; j<n; j++)
      {
        scanf("%d", &a[i][j]);
      }
    }
    for(i=0; i<m; i++)
    {
      for(j=0; j<n; j++)
```

```
    {
        sum = sum + a[i][j];
    }
}
printf("sum = %d", sum);
getch();
}
```

25/03/2017

prof. chaithra H.E

Module-3

Functions (part 1) :-

Introduction
Function prototype
Types
Function definition
Function call
Function declaration
Categories (design) of function
Actual & Formal parameters
call by value & call by reference

CHAITHRA. H.E.
B.E, M. Tech

Function :- It is a subprogram that carries out some specific & well-defined tasks.

Uses of functions

1. It reduces the code size
2. Readability of the program can be increased
3. Easier debugging
4. Function can be shared by many programmers.

If the program is very big, there are many disadvantages.

- Difficult to write large program
- very difficult to understand.

So These large programs can be divided into a series of individual related programs called module. These modules are called functions.

* Types of functions :-

1) Library (Built in) function
2) User defined function (UDF)

1) Library function :-

The C library is a collection of various types of functions which perform some standard & pre-defined tasks. These functions are called as library functions

Eg :- sqrt(); - To find square root
pow(); - To find power of number
printf(); - Send data to o/p unit
scanf(); - Accept data from keyboard

Advantages

These functions can be used whenever required.
The programmer's job is easy because the functions are already available.

Disadvantages

Since the library functions are limited, programmers can't completely rely on the library functions.

2) User Defined Function (UDF):

In most of the cases, the programmers need other than library functions to achieve some specific tasks.

The functions which are written by the user to do some specific tasks are called as User Defined Functions.

Eg:- If we want to add 2 matrices a & b, there is no library function in C to add 2 matrices. So we can write a function add_matrix(). So this function is written by user. So it is called user-defined function.

Eg:- program to add 2 numbers using functions.

```
#include<stdio.h>
int add (int a, int b);
void main()
{
    int a,b, c=0;
    a=10; b=20;
    c= add(a,b); // Function call
    printf("c is %d\n", c);
}
getch();
int add (int a, int b)
{
    int result;
```

```
    result = a+b;
    return result;
}
```

In this program the function which is present inside the main() is called as calling Function.

The function which is called by the calling function is called as called function.

When the function is called, control is transfered from main() to add(). Each statement in add() is executed & addition 2 numbers is computed.

After executing last statement i.e., return result, the control will be transfered to the main() along with result from add().

The result obtained from add() is copied to variable c. Finally result will be displayed on screen.

Elements of UDF :-

1) Function declaration
2) Function definition
3) Function call.

1) Function declaration (Function prototype)

The general syntax of function declaration is

type function_name(type a1, type a2, ---, type an);

It gives the information about return type, name of function, number of parameters.

Eg:- int add (int a, int b);

Return type is integer

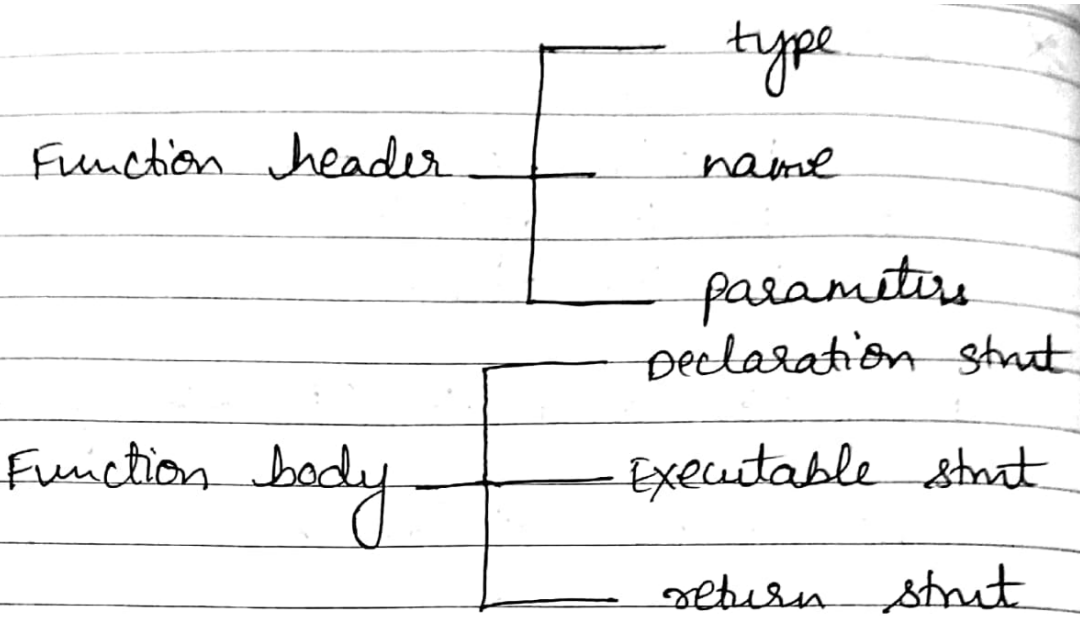
Function name is add

parameters are a & b which are of type integer.

2) Function definition :-

It consists of 2 elements -

1. Function header
2. Function body



Syntax :-

```
type name(parameters)
{
    declaration part;
    executable part;
    return part;
}
```

Eg:-

```
int add(int a, int b)
{
    int sum; // Declaration
    sum = a+b; // Executable stmt
    return sum; // return stmt
}
```

3) Function call :-

: The invokation of a function inorder to do specific task is called as function call.

Eg:-
```
void main()
{
  int a, b, sum;
  a = 10;
  b = 20;
  sum = add(a, b); // Function call
  printf("sum = %d \n", sum);
  getch();
}
```


* Actual parameter & Formal parameter

Actual parameter	Formal parameter
1) They are used in calling function. Eg:- c = sum(a,b); Here a & b are actual parameters.	1) They are used in function header of called function. Eg:- int sum(int a, int b) Here a & b are formal parameter.
2) Actual parameter can be constants, variable or expression. Eg:- c = sum(a+4, b);	2) It should be only variable. Eg:- int sum(int a, int b)

3) It sends value to Formal paramitr.	3) It recieve valuis from actual paramitr.
Eg:- c = sum(4,5);	Here, m will receive value 4 & n will take 5 val
4) Addresses of this can be sent to formal paramitr	4) If formal paramitr contains addresses, they should be declared as pointers.

*** categories (Design) of function

1) Function with no paramitr & no return type
2) Function with no paramitr & return type.
3) Function with paramitr & no returntyp.
4) Function with paramitr & returntype

1) Function with no paramitrs & no return type

* In this type, there is no data transfer between calling function & called function
* the function doesnot contain paramitrs
* The function doesnot return any value.

Eg:-
```
#include<stdio.h>
void add();
void main()
{
    add();
}
void add()
{
    int a, b, c;
    a=10;
    b=20;
    c=a+b;
    printf("sum=%d",c);
}
```


In this case the function add() do not receive any values from main() & doesnot return any value to main().

2) Function with no parameter & return type

* In this type, the function doesnot contain parameter, but function returns value.

Eg:-
```
#include<stdio.h>
int add();
void main()
{
    int c;
    c=add();
    printf("c=%d", c);
```

```
    getch();
}
int add()
{
    int a, b, c;
    a = 10;
    b = 20;
    c = a+b;
    return c;
}
```

3) Function with parameter & no return type

In this type, the function contains parameter. But it doesnot return any value.

Eg:-
```
#include<stdio.h>
void add(int a, int b);
void main()
{
    int a, b;
    a = 10;
    b = 20;
    add(a,b);
}
void add(int a, int b)
{
    int c;
```

```
    c = a+b;
    printf("c=%d", c);
}
```

4) Function with parameter & return type

* In this type, function contains both parameter & return type.

Eg:-
```
#include<stdio.h>
int add(int a, int b);
void main()
{
    int a, b, c;
    a=10;
    b=20;
    c= add(a,b);
    printf("c=%d", c);
}
int add(int a, int b)
{
    int c;
    c= a+b;
    return c;
}
```

(Passing parameters to function)

** Call by value & call by reference

1) call by value :- (pass by value)

In this type, function is called with actual parameter, where values of actual parameter will be copied to formal parameter.

Eg:- program to add 2 numbers

```
#include<stdio.h>
void add(int m, int n);
void main()
{  int a, b;
   a=10; b=20;
   add(a,b);
   getch();
}
void add(int m, int n)
{  int c;
   c=m+n;
   printf("c=%d", c);
}
```


2) call by reference :- (pass by reference)

In this type, function is called with address of actual parameter, where addresses of actual parameters are passed

to formal parameter

Eg:-

```
#include <stdio.h>
void add (int *m, int *n);
void main()
{
    int a, b;
    a=10; b=20;
    add(&a, &b);
}
void add (int *m, int *n)
{
    int c;
    c = *m + *n;
    printf("c=%d", c);
}
```

pass by value	pass by reference
1) when a function is called, the values of variables are passed	1) Here, addresses of variables are passed
2) Execution is slower since all the values have to be copied to Formal parameter	2) Execution is faster since only addresses are copied
3) change of formal parameter will not affect actual parameter	3) Actual parameter are changed since formal parameter indirectly manipulates actual parameter

passing array to function

In this, array is passed as a parameter to function.

* WAP to read & print n array elements using function by passing array as a parameter.

```
#include<conio.h>
#include<stdio.h>  → int i,n; //global declarati
void input(int a[10]);
void output(int a[10]);
void main()
{
 int x[10];
 clrscr();
 printf("Enter n\n");
 scanf("%d",&n);
 input(x);
 output(x);
 getch();
}
void input(int a[10])
{
 printf("Enter n elements\n");
 for(i=0; i<n; i++)
 scanf("%d",&a[i]);
}
void output(int a[10])
{
 printf("output array is\n");
```

Prof. Chaithra H.E

```
for(i=0; i<n; i++)
printf("%d\t", a[i]);
}
```

Note :- similarly try to write bubble sort program by passing array as a parameter to function.

Module 3 (cont....)

Recursion (part 2)

CHAITHRA. H.E
B.E, M.Tech

Introduction to recursion.
How it works?
Fibonacci using recursion
Factorial using recursion

Recursion :- A function that calls itself is called Recursive function & this technique is called Recursion.

How it works?

```
void recurse();
void main()
{
  recurse();
}

void recurse()
{
  recurse();
}
```

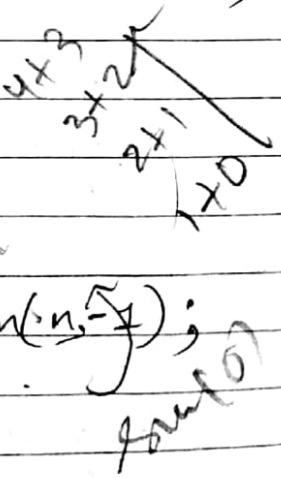
In the above example the function recurse() which is present in main() calls function recurse(). & further recurse() calls the function recurse() which is present inside function definition.

The recursion continues until some condition is met.

To prevent infinite recursion, if-else stmt can be used where one branch makes recursive call & other doesnot.

program to find sum of natural numbers using recursion.

```
int sum(int n);
void main()
{
  int n, result;
  printf("Enter n\n");
  scanf("%d", &n);
  result = sum(n);
  printf("Sum= %d", result);
}
int sum(int n)
{
  if(n!=0)
  return n + sum(n-1);
  else
  return n;
}
```



** WAP to find factorial of a given number using recursive function.

```
#include <stdio.h>
int n, fact = 1, result;
int factorial (int n);

void main()
{
  clrscr();
  printf("Enter n\n");
  scanf("%d", &n);
  result = factorial(n);
  printf("Factorial = %d", result);
}

int factorial (int n)
{
  if (n==1)
  return 1;
  else
  fact = n * factorial(n-1);
  return fact;
}
```

** WAP to generate fibonacci series using recursion.

```
int fib(int n);

void main()
{
  int i, n, result;
```

```
clrscr();
printf("Enter n\n");
scanf("%d", &n);
for(i=0; i<n; i++)
result = fib(i);
printf("%d\t", result);
}

int fib(int n)
{
  if(n<2)
  return n;
  else
  return (fib(n-1) + fib(n-2));
}
```

* Lab program 8 (isprime)