Arrays Functions Storage classes& Pointers

MODULE 2 COMPLETE AND MODULE 3 INTRO

Arrays

int a;

a=1,2,3,4; X

→Multiple values cannot be stored in a normal integer variable.

To store Multiple values under a common variable name we use the concept of array.

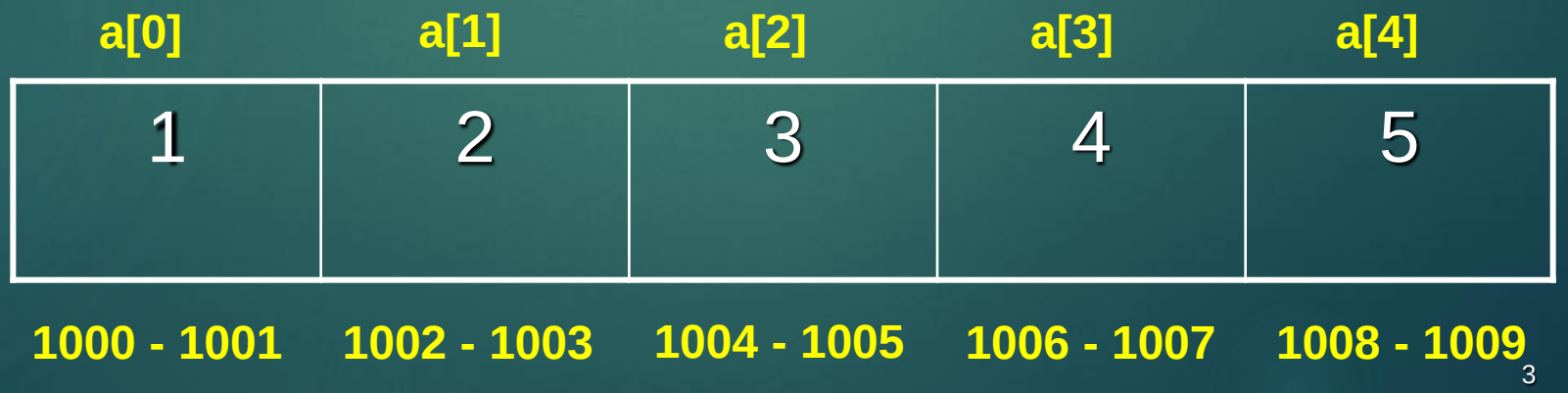
Arrays

Definition:

→An Array is a collection of similar data type under a common variable name.

→The elements in an array are stored in consecutive or sequential memory location

→Dissimilar data items cannot be stored in an array



Types of Array

1.One Dimensional Array.(1-D)

2.Two Dimensional Array.(2-D)

3.Multi Dimensional Array.

Arrays(1-D) - Declaration

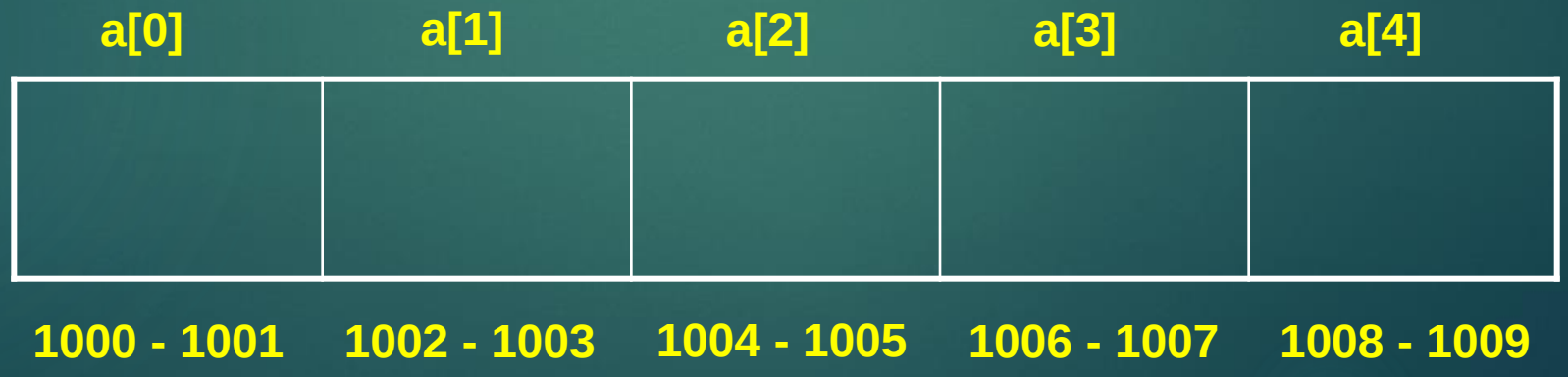
Syntax:

Data type variable name [size];

Eg:

int a[5];

a→array of type int capable of holding max 5 integers.



Arrays - Initialization

- An array can be initialized either during Compile (Static) time or Run (Dynamic) Time.

Compile Time Assignment:

int a[5] = { 1,2,3,4,5 } int a[ ] = { 1,2,3,4,5 }

Char b[5] = { 'A','B','C','D','E' }

a[0]	a[1]	a[2]	a[3]	a[4]
1	2	3	4	5
1000 - 1001	1002 - 1003	1004 - 1005	1006 - 1007	1008 - 1009

b[0]	b[1]	b[2]	b[3]	b[4]
A	B	C	D	E
1000	1001	1002	1003	1004

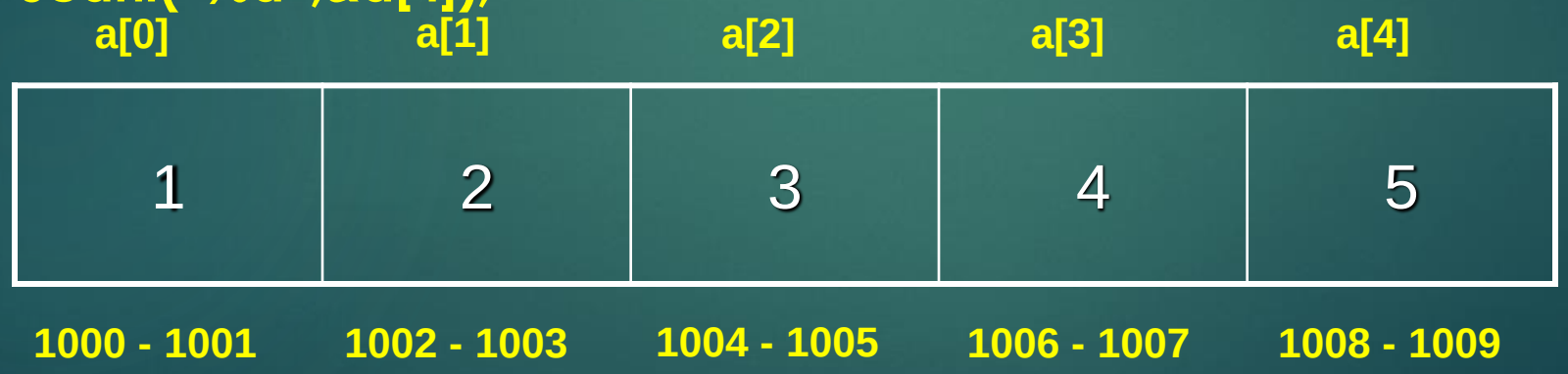
Arrays - Initialization

- An array can be initialized either during Compile (Static) time or Run (Dynamic) Time.

Run Time Assignment:(i/p through scanf Statement)

```
int a[5];
Scanf("%d",&a[0]);
Scanf("%d",&a[1]);
Scanf("%d",&a[2]);
Scanf("%d",&a[3]);
Scanf("%d",&a[4]);
```

```
for(i=0;i<5;i++)
{
    scanf("%d",&a[i]);
}
```



Example


```
#include<stdio.h>
#include<conio.h>
Void main( )
{
int a[5] = { 1,2,3,4,5 };
int b[5];
Printf("Enter the values for array b");
For(i=0;i<5;i++)
    Scanf("%d",&b[i]);//Read b
For(i=0;i<5;i++)
    printf("%d",a[i]); //print a
For(i=0;i<5;i++)
    printf("%d",b[i]); //print b
```

Output:

Enter the values for array b

10 20 30 40 50

1 2 3 4 5

10 20 30 40 50

Here is a program that prints out the memory locations in which the elements of this array are stored.

```
main( )
{
int num[ ] = { 24, 34, 12, 44, 56, 17 } ;
int i ;
for ( i = 0 ; i <= 5 ; i++ )
{
printf ( "\nelement no. %d ", i ) ;
printf ( "address = %u", &num[i] ) ;
}
}
```

The output of this program would look like this:

element no. 0 address = 65512

element no. 1 address = 65514

element no. 2 address = 65516

element no. 3 address = 65518

element no. 4 address = 65520

element no. 5 address = 65522

[A] What would be the output of the following program:


```
main( )
{
int num[26], temp ;
num[0] = 100 ;
num[25] = 200 ;
temp = num[25] ;
num[25] = num[0] ;
num[0] = temp ;
printf ( "\n%d %d", num[0], num[25] ) ;
}
```

[B] What would be the output of the following program:

```
main( )
{
int array[26], i ;
for ( i = 0 ; i <= 25 ; i++ )
{
array[i] = 'A' + i ;
printf ( "\n%d %c", array[i], array[i] ) ;
}
}
```

[C] What would be the output of the following program:

```
main( )
{
int mark[50], i ;
for ( i = 0 ; i <= 48 ; i++ )
{
mark[i] = i ;
printf ( "\n%d", mark[i] ) ;
}
}
```

A Simple Program Using Array

Let us try to write a program to find average marks obtained by a class of 30 students in a test.

```
main( )
{
int avg, sum = 0 ;
int i ;
int marks[30] ; /* array declaration */
for ( i = 0 ; i <= 29 ; i++ )
{
printf ( "\nEnter marks " ) ;
scanf ( "%d", &marks[i] ) ; /* store data in array */
}
for ( i = 0 ; i <= 29 ; i++ )
sum = sum + marks[i] ; /* read data from an array*/
avg = sum / 30 ;
printf ( "\nAverage marks = %d", avg ) ;
}
```

Bubble Sorting

```
main()
{
    int n,a[20],i,j,t;
    printf("Enter the number of elements in the array:");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
            printf("\nEnter the a[%d] number = ",i);
            scanf("%d",&a[i]);
    }
    for(i=0;i<n;i++)
       for(j=i+1;j<n;j++)
          if(a[i]>a[j])
          {
                t=a[i];
                a[i]=a[j];
                a[j]=t;
          }
    printf("\nThe sorted array elements are:");
    for(i=0;i<n;i++)
                printf("\nElement a[%d] number = %d",i,a[i]);
       getch();
}
```

```
#include<stdio.h>
#include<conio.h>
main()
{
    int f,l,mid,n,a[20],x,i;
    printf("Enter the number of elements in the array:");
    scanf("%d",&n);
    for(i=0;i<n;i++)
    {
        printf("\nEnter the a[%d] number = ",i);
        scanf("%d",&a[i]);
    }
    printf("\nEnter the element to be found:");
    scanf("%d",&x);
    f=0;
    l=n-1;
}
while(f<=l)
    {
        mid=(f+l)/2;
        if(a[mid]==x)
        {
            printf("\nThe Element %d is found in %d location",x,mid+1);
            getch();
            exit(0);
        }
        else if(x>mid)
            f=mid+1;
        else if(x<mid)
            l=mid-1;
    }
    printf("\nThe element %d is not found",x);
    getch();
}
```

```
#include<stdio.h>
main()
{
int a[10],n;
printf("Enter the size of array:");
scanf("%d",&n);
printf("Enter the Array elements:");
for(i=0;i<n;i++)
      {
        printf("\nenter the a[%d] element:",i);
        scanf("%d",&a[i]);
      }
for(i=0;i<n;i++)
      {
        printf("\nThe element a[%d] is %d:",i,a[i]);
      }
getch();
}
```

2 – D Array

Syntax:

Data_type variable_name [Row_Size][Column_Size];

Eg:

int a[3][3];

- a→array of type int capable of holding max 3 Rows and 3 Columns of Integer Values.
- Row major ordering is followed.

	0th Col	1st Col	2nd Col
0th Row	A[0][0]	A[0][1]	A[0][2]
1st Row	A[1][0]	A[1][1]	A[1][2]
2nd Row	A[2][0]	A[2][1]	A[2][2]

Initialize the 2-D Array

Syntax:

```
Data_Type Variable_Name[ ][ ] = {{Row_Wise_Value}…};
```

Ex:

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

or

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

	0th Col	1st Col	2nd Col
0th Row	1	2	3
1st Row	4	5	6
2nd Row	7	8	9

Declaring 2-D Array


```
#include<stdio.h>
#include<conio.h>
void main( )
{
int a[ ][ ] = { {1,2},{4,5} };
int b[3][3];
}
```

```
#include<stdio.h>
main()
{
int a[3][3],n,i,j;
printf("Enter the size of array:");
scanf("%d",&n);
printf("Enter the Array elements:");
for(i=0;i<n;i++)
  for(j=0;j<n;j++)
      {
        printf("\nenter the a[%d][%d] element:",i,j);
        scanf("%d",&a[i][j]);
      }
for(i=0;i<n;i++)
    for(j=0;j<n;j++)
        printf("\nThe element a[%d][%d] is %d:",i,j,a[i][j]);
getch();
}
```

Example


```
#include<stdio.h>
#include<conio.h>
void main( )
{
int a[][] = { {1,2},{4,5} };
int b[3][3];
printf("Enter the values for array 'B' ");
for(i=0;i<3;i++)
  for(j=0;j<3;j++)
        scanf("%d",&b[i][j]);//Read b
printf("The Value of 'A' Array\n");
for(i=0;i<2;i++)
{
 for(j=0;j<2;j++)
        printf("%d\t",a[i][j]); //print a
 printf("\n");
}
printf("The Value of  'B' Array");
for(i=0;i<3;i++)
{
 for(j=0;j<3;j++)
        printf("%d\t",b[i][j]); //print b
 printf("\n");
}
```

Output:

Enter the values for array 'B'
10 20 30
10 20 30
10 20 30

The Value of 'A' Array
1 2
4 5

The Value of 'B' Array

10	20	30
10	20	30
10	20	30

Functions

Types of Functions:

1.Inbuilt or Library Function:

Functions which are already available or predefined in the compiler are said to be inbuilt or Library Functions

2.User Defined Function.

Functions that are defined by the user are said to be user defined function.

String Manipulating Functions

1.String Length.

2.String Copy.

3.String Concatenation.

4.String Compare.

5.String Reversal.

All string Manipulating Functions are stored in String.h header file.

String Length

1.String Length – Is used to find the Length of a string.

Syntax:

```
int variable = strlen( string variable )
```

Eg:

```
char name[10];
int a;
name = gets( );//scanf("%s",name);
a = strlen( name );
printf("%d",a);
```

output:

```
God
3
```

String Copy

1.String copy – String copy function is used to copy the content of one string to another.

Syntax:

strcpy(s1 , s2);

Content of s2 will be copied to s1

S2 – Source

S1 – Destination

Eg:

```
char s1[10],s2[10]="program";          strcpy(s1,s2);
output:
puts(s1);                              program
puts(s2);                              program
```

String Concatenation

1.String concatenation – The function is used to concatenate (Combining Two strings together) the content of one string to another.

Syntax:

strcat(s1 , s2);

Content of s2 is cocatenated with s1

S1 – Destination contains the concatenated string

Eg:

```
char s1[10]="Comp";
char s2[10]="uter";          strcpy(s1,s2);          output:
puts(s1);                    computer
```

String Reversal

1.String Reversal – String reversal function is used to reverse a given string.

Syntax:

strrev(s1);

S1 → String to be reversed.

Eg:

```
char s1[10]="Test";          strrev(s1);          output:
puts(s1);                    tseT
```

String Compare

1.String compare – String compare function is used to compare two strings for equality.

Syntax:

```
strcmp( s1 , s2 );
```

→Corresponding characters(Ascii) in s1 and s2 are compared. After comparing if all the corresponding character in s1 and s2 are equal then the function returns 0 (s1 and s2 are equal)

→The function returns the ascii difference of dissimilar character if both strings are not equal.

Example

Case(i)

S1 = TEST

S2 = TEST

→In both s1 and s2 all the characters are equal hence the function strcmp( ) returns zero (0).[String s1 & s2 are equal]

Case(ii)

S1 = Anil [Ascii of A = 65]

S2 = anil [Ascii of a = 97]

→A and a are dissimilar characters hence the function strcmp( ) returns their ascii difference which is negative, hence returns -1(string s2 is greater than s1)

a)Output


```
main( )
{
char name[ ] = “VIT-Chennai" ;
int i = 0 ;
while ( i <= 7 )
{
printf ( "%c", name[i] ) ;
i++ ;
}
```

b)Output


```
main( )
{
char name[ ] = “VIT-Chennai" ;
int i = 0 ;
while ( name[i] != ‘\0’ )
{
printf ( "%c", name[i] ) ;
i++ ;
}
}
```

Inbuilt Library Functions

1.I / O Manipulating Functions:

1.printf( ) 2.Scanf( ) 3.getchar( ) 4.putchar( ) 5.gets( ) 6.puts( )

2.Character Testing Functions:

1.isupper( ) 2.islower( ) 3.toupper( ) 4.tolower( )

3.String Functions:

4.Mathematical Functions

5.File Handling Functions.

Refer your book for more In-built functions.

User Defined Functions

→Functions that are defined by the user are said to be user defined function.

Terms used in Functions:

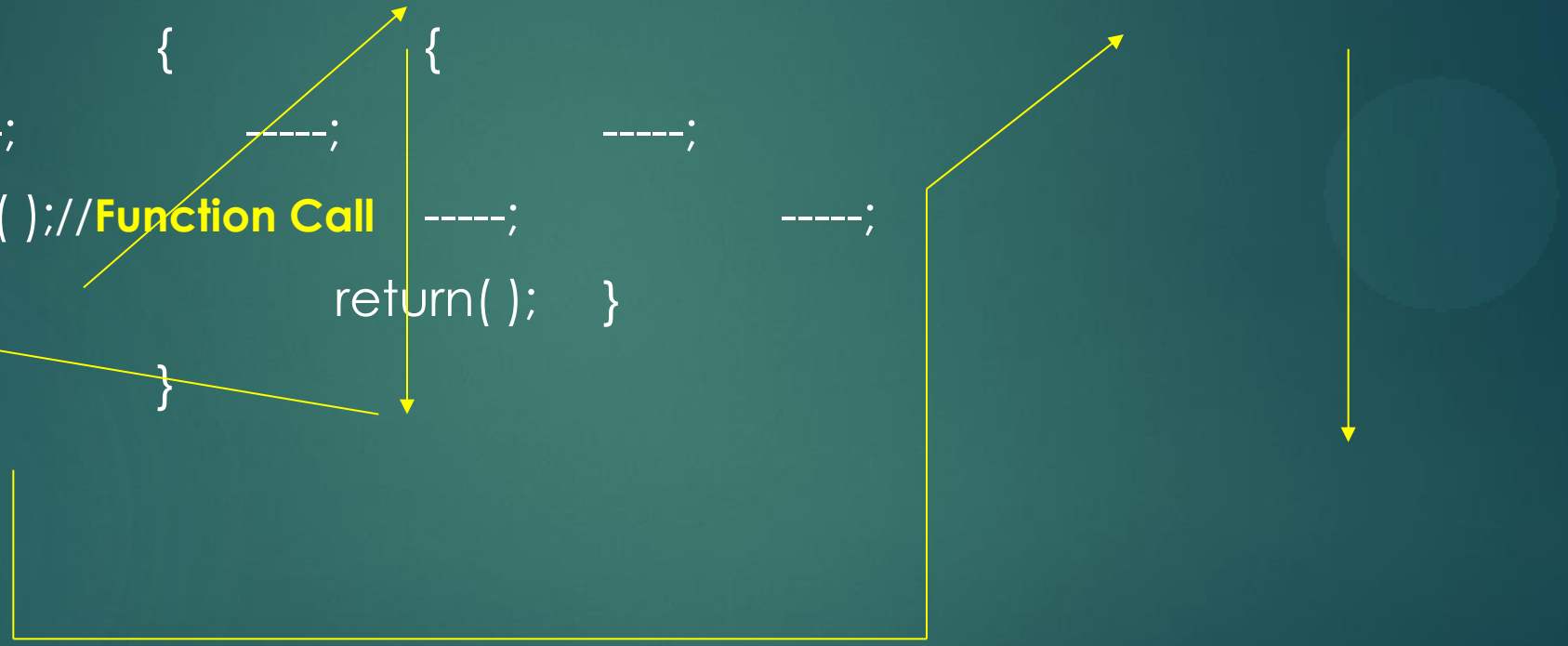
1. Function Declaration or Prototype.

2. Function Call.

3. Function Definition.

Example


```
float f1( );   //Declaration or prototype
Void f2( );
Void main( )    float f1( )        void f2( )(Definition)
{               {                  {
-------;        -----;             -----;
X=f1( );//Function Call  -----;              -----;
f2( );                  return( );  }
                }
}
```



User Defined Function

Function Prototypes:

1. Function without arguments and without return type.
2. Function without arguments and with return type.
3. Function with arguments and without return type.
4. Function with arguments and with return type.

Arguments or Parameters

→ The variables that are passed in a function are said to be arguments or Parameters.

Eg:

```
void main( )          f1( int  x, int  y)
{                     {
int a,b,z;                  int c;
z=f1( a, b);          c=a+b;
printf("%d",z);             return( c );
}                     }
```

Where a & b are Actual Parameters

Where x & y are Formal Parameters

→The memory Address of Actual Parameter and formal Parameters are Different.

1.Function without arguments and without return type.


```
Void swap( );
Void main( )
{
   swap( );
   //Function call;
}
```

```
void swap( ) //Definition
{
   int a,b,t;
   printf("Enter a & b:");
   scanf("%d %d",&a,&b);
   t = a;
   a = b;
   b = t;
   printf("Swapped Values:");
   printf("%d %d",a,b);
}
```

2.Function without arguments and with return type.


```
int add( );
Void main( )
{
    int z;
    z=add( );
    printf("Sum=%d",z);
}
```

```
void add( ) //Definition
{
    int a,b,c;
    printf("Enter a & b:");
    scanf("%d %d",&a,&b);
    c=a+b;
    return( c );
}
```

3.Function with arguments and without return type.


```
Void add(int ,int );
Void main( )
{
    int a,b;
    printf("Enter a & b:");
    scanf("%d %d",&a,&b);
    add(a,b );
}
```

```
void add(int x, int y ) //Definition
{
    int c;
    c=x+y;
    printf("sum=%d",c);
}
```

Where a & b in main( ) are actual Parameters.

X & y in add( ) are Formal Parameters.

4.Function with arguments and with return type.

```
int add(int ,int );
Void main( )
{
    int a,b,z;
    printf("Enter a & b:");
    scanf("%d %d",&a,&b);
    z=add(a,b );
    printf("Sum=%d",z);
}

int add(int x, int y ) //Definition
{
    int c;
    c=x+y;
    return( c  );
}
```

Where a & b in main( ) are actual Parameters.

X & y in add( ) are Formal Parameters.

Examples

1. Write a function in ‘C’ to perform factorial of ‘n’ natural numbers.

2. Write a function using ‘C’ to find the square of a number.

3. Write a ‘C’ Program to perform Matrix Addition for n x n Matrix.

Selection sort

i/p :

o/p :

For Selection sort apply the logic of Max and Min of an array.

Selection sort

i/p :

Min=a[0] = 5 ; compare a[0] with all other elements till n-1.

(i) if(5 > 4) then min=4

(ii) if(4>0) then min=0

(iii) if(0>2)

(iv) if(0>1)

Final value of Min = 0

Interchange the min value in a[2] with a[0]

Selection sort

i/p :

Min=a[1] = 4 ; compare a[1] with all other elements till n-1.

(i) If(4 >5)

(ii) if(4>2) then min=2

(iii) if(2>1) then min=1

Final value of Min = 1 ; a[4]

Interchange the min value in a[4] with a[1]

```
for(i=0;i<n;i++)
    {
    min=a[i];
    for(j=i+1;j<n;j++)
    {
                if(min>a[j])
                {
                        min=a[j];
                        k=j;
                }
    }
    t=a[i];
    a[i]=min;
    a[k]=t;
    }
```

Example

1. Write a ‘C’ Program to sort the array elements using Bubble and Selection sort.
2. Write a ‘C’ Program to Eliminate the duplicate elements from the array.

i/p : 1 2 2 3 4 5 6 3 (n=8)

o/p : 1 2 3 4 5 6 (n=6)

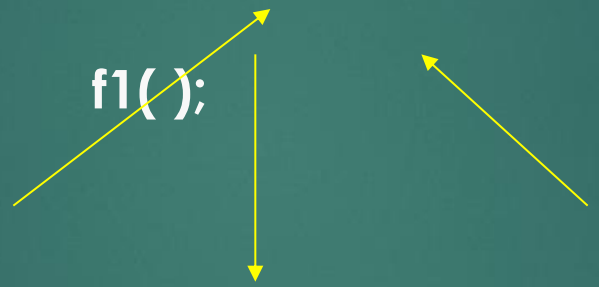
3. Write a ‘C’ program to perform Matrix addition for n x n Matrix

Recursive Function or Recursion

→A Function which calls the same function itself again and again until some condition is satisfied is said to be **Recursive function**

Eg:

```
main( )        f1( )
{              {
    f1( );         f1( );
}              }
```



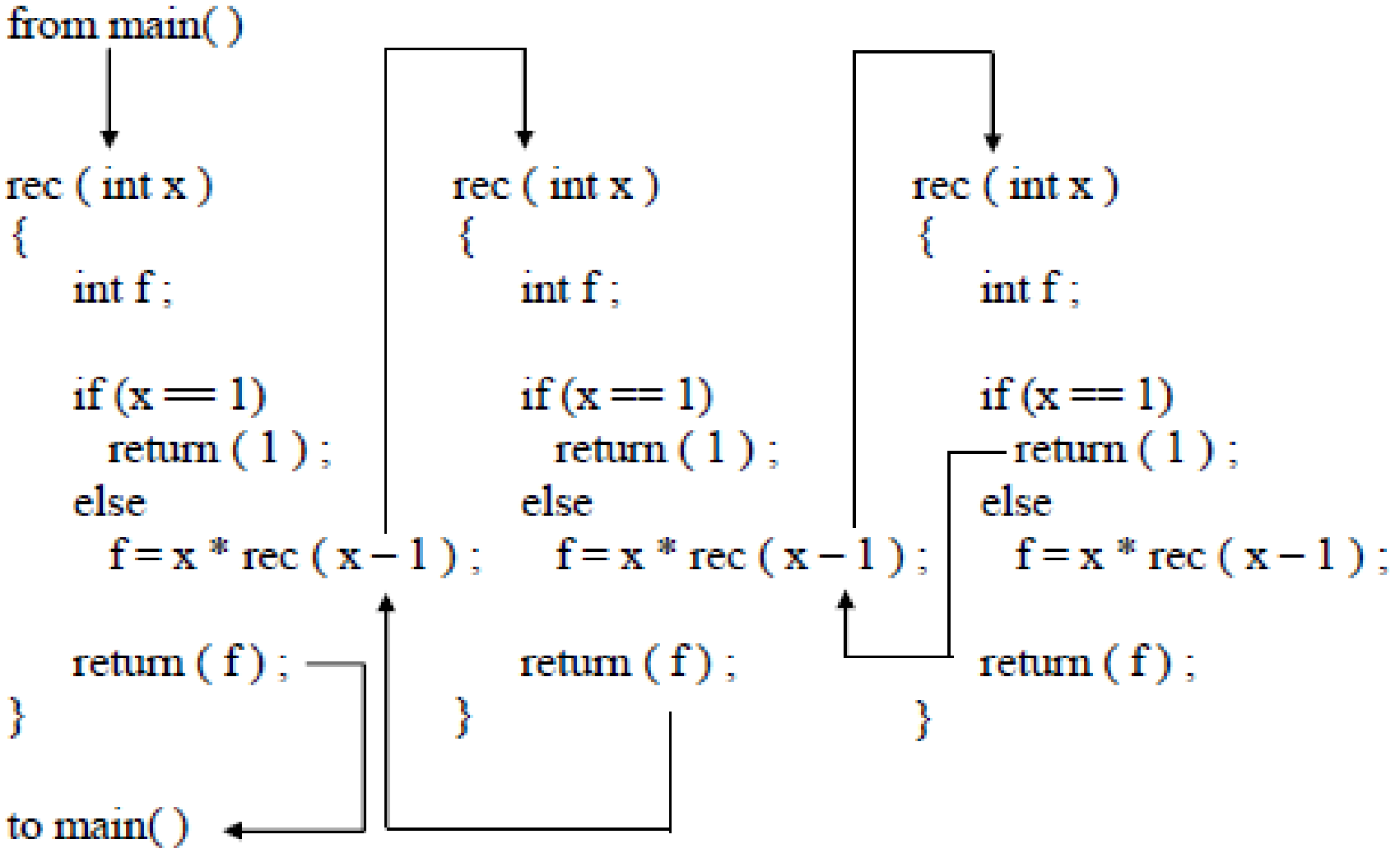
```
//Factorial using recursion
#include<stdio.h>
#include<conio.h>
int rec(int);
void main()
{
        int n,z;
        clrscr( );
        printf("Enter the value of n:");
        scanf("%d",&n);
        z=rec(n);
        printf("\nFactorial of %d is %d",n,z);
        getch();
        }
        int rec(int x)
        {
        int f;
        if(x==0 || x==1)
            return(1);
        else
        {
            f=x*rec(x-1);
            return(f);
        }
  }
```

i/p: Enter the value of n: 3

Call1:
x=3
goes to else part
f = 3 * rec(2);
Return(f) is pending

Call2:
x=2
Goes to else part
f = 2 * rec(1);
Return(f) is pending

Call3:
x=1
If(x==0||x==1) is true
Return(1);


from main( )
rec (int x)
{
int f ;
if (x == 1)
return (1) ;
else
f = x * rec (x – 1) ;
return (f) ;
}
to main( )
rec (int x)
{
int f ;
if (x == 1)
return (1) ;
else
f = x * rec (x – 1) ;
return (f) ;
}
rec (int x)
{
int f ;
if (x == 1)
return (1) ;
else
f = x * rec (x – 1) ;
return (f) ;
}

```
#include<stdio.h>
#include<conio.h>
void reverse();
main()
{
      reverse();
      getch();
}
void reverse()
{
     char c;
     while(c=getchar()!='\0')
     {
      reverse();
      putchar(c);
     }
}
```

i/p:
Test

Output:
tseT

Multiple Function calls: Example


```
#include<stdio.h>
#include<conio.h>
Void f1( );
Void main( )
{
    -----;
    f1( );
    f1( );
}
```

```
void f1( )
{
 int a=1;
      a++;
printf("%d",a);
}
```

Output:

2

2

Example


```
#include<stdio.h>
#include<conio.h>
Void f1( );                 void f1( )
Void main( )                {
{                           static int a=1;
    -----;                        a++;
    f1( );                  printf("%d",a);
    f1( );              }
}
```

Output:

2

3

```
// Program to calculate the sum of array elements by
passing to a function
#include <stdio.h>
float calculateSum(float age[]);
int main() {
float result, age[] = {23.4, 55, 22.6, 3, 40.5, 18};
// age array is passed to calculateSum()
result = calculateSum(age);
printf("Result = %.2f", result);
return 0;
}
float calculateSum(float age[]) {
float sum = 0.0;
for (int i = 0; i < 6; ++i) {
sum += age[i];
}
return sum;
}
```

```
#include <stdio.h>
void displayNumbers(int num[2][2]);
int main()
{
int num[2][2];
printf("Enter 4 numbers:\n");
for (int i = 0; i < 2; ++i)
for (int j = 0; j < 2; ++j)
scanf("%d", &num[i][j]);
// passing multi-dimensional array to a function
displayNumbers(num);
return 0;
}
void displayNumbers(int num[2][2])
{
printf("Displaying:\n");
for (int i = 0; i < 2; ++i)
for (int j = 0; j < 2; ++j)
printf("%d\n", num[i][j]);
}
```

Storage Classes

MODULE 2 PART B

Storage Class

- Storage class refers to the permanence of a variable and its scope (visibility of a variable with in a program).
- Variables in C are handled differently by the compiler based on the type of storage class used.

The Storage class tells us:

→Where the variable would be stored

→The scope of the variable.

→What will be the initial value of the variable.

Storage Class

Types of Storage Class

1. Automatic variable (Auto)

2. External Variables (extern)

3. Static variable (static)

4. Register variable (Register)

Storage Class

Automatic Variable: (Local Variables)

→The Variables that are declared with in the function is said to be Automatic or Local Variable.

→The features of a variable defined to have an automatic storage

class are as under:

Storage − Memory.

Default initial value − An unpredictable value, which is often called a garbage value.

Scope − Local to the block in which the variable is defined.

Life − Till the control remains within the block in which the variable is defined.

Storage Class

Automatic Variable:

```
main( )
{
    int a,b; // Automatic variable
    --------;
}

main( )
{
    auto int a,b;
    ---------------;
}
```

```
main( )
{
auto int i = 1 ;
{
    {
     {
          i++;
          printf ( "\n%d ", i ) ;
     }
     printf ( "%d ", i ) ;
    }
    printf ( "%d", i ) ;
}
}
```

The output of the above program is:

2 1 1

```
main( )
{
auto int i = 1 ;
    {
     auto int i = 2 ;
     {
          auto int i = 3 ;
          printf ( "\n%d ", i ) ;
     }
     printf ( "%d ", i ) ;
    }
printf ( "%d", i ) ;
}
```

The output of the above program would be:

3 2 1

Storage Class

External Variable: (Global Variable)

→The Variables that are declared outside the function is said to be External or Global Variable.

→These variables can be used by all the functions in a program. They are alive and active throughout the entire program.

→The scope of External variable is Global.

External Variable

The features of a variable whose storage class has been defined as

external are as follows:

Storage – Memory.

Default initial value – Zero.

Scope – Global.

Life – As long as the program's execution doesn't come to an end.

Storage Class

External Variable:

```
extern int a,b; // External variable
main( )
{
      int a,b; // Automatic variable
      --------;
}
```

Note: The Key word Auto and Extern are optional.

External Variable

Examples:

```
#include<stdio.h>
int a=10; // Global variable (External)
void main( )
{
    int a=20; // Local Variable (Auto)
    printf("%d",a);
    f1( );
}
f1( )
{
    printf("%d",a);
}
```

Output:

20
10

```
int i ;
main( )
{
printf ( "\ni = %d", i ) ;
increment( ) ;
increment( ) ;
decrement( ) ;
decrement( ) ;
}
increment( )
{
i = i + 1 ;
printf ( "\n on incrementing i =
    %d", i ) ;
}
decrement( )
{
i = i - 1 ;
printf ( "\non decrementing i = %d", i )
    ;
}
```

The output would be:

```
i = 0
on incrementing i = 1
on incrementing i = 2
on decrementing i = 1
on decrementing i = 0
```

Storage Class

Static Variable:

→The Variables that are declared as static are permanent within their own functions.

→Static variables are declared within a function with a key word static.

Example


```
#include<stdio.h>
#include<conio.h>
Void f1( );                    void f1( )
Void main( )                   {
{                            int a=1;
   -----;                        a++;
   f1( );                        printf("%d",a);
   f1( );                  }
}
```

Output:

2

2

Example


```
#include<stdio.h>
#include<conio.h>
Void f1( );
Void main( )
{
   -----;
   f1( );
   f1( );
}
```

```
void f1( )
{
static int a=1;
   a++;
   printf("%d",a);
}
```

Output:

2

3

Static Variable

The features of a variable defined to have a **static storage class are** as under:

Storage − Memory.

Default initial value − Zero.

Scope − Local to the function in which the variable is defined.

Life − Value of the variable persists between different function calls.

```
main( )
{
    increment( ) ;
    increment( ) ;
    increment( ) ;
}

increment( )
{
    auto int i = 1 ;
    printf ( "%d\n", i ) ;
    i = i + 1 ;
}
```

```
main( )
{
    increment( ) ;
    increment( ) ;
    increment( ) ;
}

increment( )
{
    static int i = 1 ;
    printf ( "%d\n", i ) ;
    i = i + 1 ;
}
```

The output of the above programs would be:

```
1
1
1
```

```
1
2
3
```

Register Storage Class

The features of a variable defined to be of **Register storage class** are as under:

Storage - CPU registers.

Default initial value - Garbage value.

Scope - Local to the block in which the variable is defined.

Life - Till the control remains within the block in which the variable is defined.

Example

A value stored in a CPU register can always be accessed faster than the one that is stored in memory.

A good example of frequently used variables is

loop counters. We can name their storage class as **register.**

```
main( )
{
register int i ;
for ( i = 1 ; i <= 10 ; i++ )
printf ( "\n%d", i ) ;
}
```

Type def

→Type def is a user defined data type which is used to rename an existing data type.

```
typedef int mark;
typedef int length;
mark m1,m2;
length l1,l2;
```

Pointers

MODULE3 INTRODUCTION

Pointers


```
int  a=1,b;
```

1000 – 1001

1002 – 1003

```
b = a; // b=1;
```

```
b = &a; X
```

&a = 1000

→Address can only be stored in pointer variable.

Pointers

Definition:

→A variable which is capable of holding address of another variable is said to be a pointer variable.

→Using pointer variable it is easier for accessing memory location of another variable.

Pointers - Declaration

Syntax:

Data Type * variable;

Where

data type → Type of variable whose address is to be stored in pointer variable.

* → Value at address operator or pointer operator.

Pointers


```
int  a=1;
int  *b; // b is a pointer variable
```

b = &a; &a = 1000 ; b=1000

b is pointing to address location 1000

*b = * (1000) = 1

→Using pointers it is possible to access value of another variable by storing its address.

Parameter passing Methods

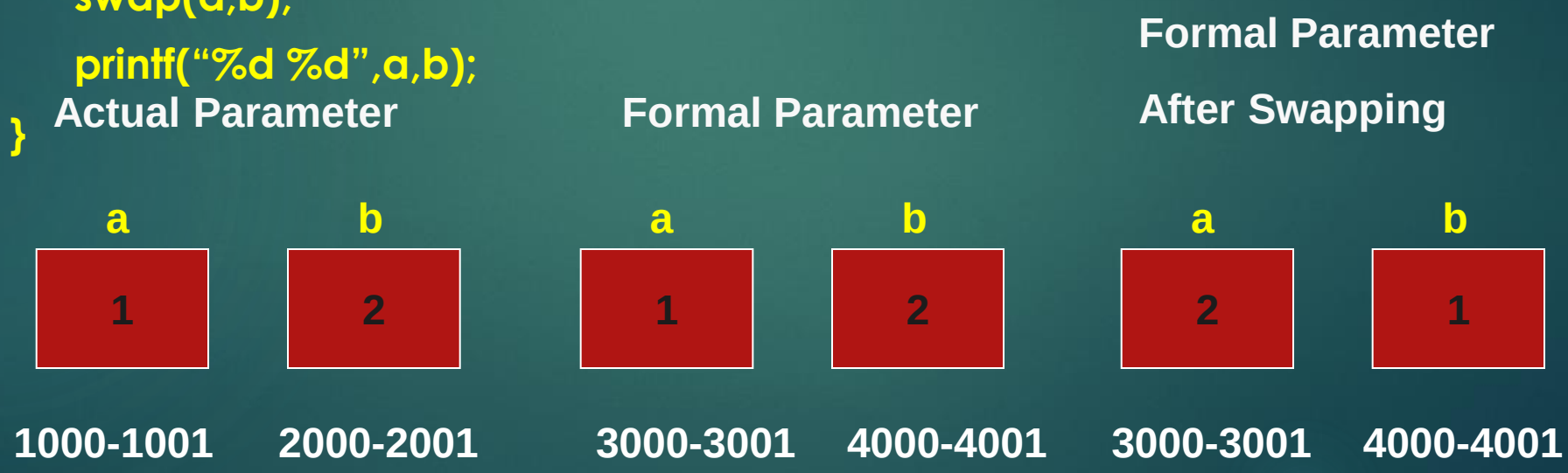
1.Call by value

2.Call by Reference. (Using Pointers)

Call By Value


```
#include<stdio.h>
#include<conio.h>
Void swap(int,int);
Void main( )
{
    int a,b;
    printf("Enter a & b:");
    Scanf("%d %d",&a,&b);
    swap(a,b);
    printf("%d %d",a,b);
}
```

```
void swap( int a, int b)
{
    int t;
    t = a;
    a = b;
    b = t;
    printf("%d %d",a,b);
}
```



Call By Value

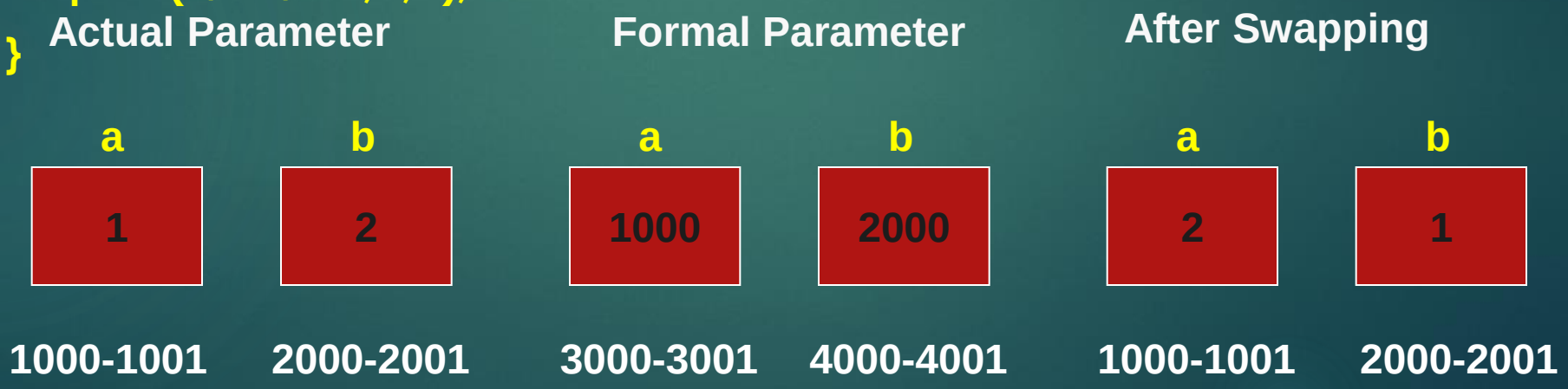
→The values of Actual Parameter is Copied to the corresponding Formal Parameter.

→The changes done within function will affect only the Formal Parameter. Since the address of actual parameter is different from formal the values of actual parameter remains the same.

Call By Reference

```
#include<stdio.h>
#include<conio.h>
Void swap(int *,int *);
Void main( )
{
    int a,b;
    printf("Enter a & b:");
    Scanf("%d %d",&a,&b);
    swap(&a,&b);
    printf("%d %d",a,b);
}
```

```
void swap( int *a, int *b)
{
        int t;
        t = *a;
        *a = *b;
        *b = t;
printf("%d %d",*a,*b);
}
```



```
main( )
{
int i = 3 ;
int *j ;
j = &i ;
printf ( "\nAddress of i = %u", &i ) ;
printf ( "\nAddress of i = %u", j ) ;
printf ( "\nAddress of j = %u", &j ) ;
printf ( "\nValue of j = %u", j ) ;
printf ( "\nValue of i = %d", i ) ;
printf ( "\nValue of i = %d", *( &i ) ) ;
printf ( "\nValue of i = %d", *j ) ;
}
```

Address i = 1000-1003

Address j = 2000-2002

OUTPUT → ??

```
main( )
{
int radius ;
float area, perimeter ;
printf ( "\nEnter radius of a circle " ) ;
scanf ( "%d", &radius ) ;
areaperi ( radius, &area, &perimeter ) ;
printf ( "Area = %f", area ) ;
printf ( "\nPerimeter = %f", perimeter ) ;
}
areaperi ( int r, float *a, float *p )
{
*a = 3.14 * r * r ;
*p = 2 * 3.14 * r ;
}
```

And here is the output...

```
Enter radius of a circle 5
Area = 78.500000
Perimeter = 31.400000
```

1.OUTPUT: ??

```
main( )
{
int i = 5, j = 2 ;
junk ( &i, &j ) ;
printf ( "\n%d %d", i, j ) ;
}
junk ( int *i, int *j )
{
*i = *i * *i ;
*j = *j * *j ;
}
```

2.OUTPUT: ??

```
main( )
{
float a = 13.5 ;
float *b, *c ;
b = &a ; /* suppose address of a is 1006 */
c = b ;
printf ( "\n%u %u %u", &a, b, c ) ;
printf ( "\n%f %f %f %f %f", a, *(&a), *&a, *b, *c )
    ;
}
```

Output

```c
#include<stdio.h>

main( )
{
float a = 13.5 ;
float *b, *c ;
b = &a ; /* suppose address of a is 1006 */
c = b ;
printf ( "\n%u %u %u", &a, b, c ) ;
printf ( "\n%0.2f %f %f %f %f", a, *(&a), *&a, *b, *c ) ;
}
```

```
/tmp/gcZE7wBTlO.o
2917646004 2917646004 2917646004
13.50 13.500000 13.500000 13.500000 13.500000
```

```
main( )
{
int i ;
int marks[ ] = { 55, 65, 75, 56, 78, 78, 90 } ;
for ( i = 0 ; i <= 6 ; i++ )
display ( marks[i] ) ;
}
display ( int m )
{
printf ( "%d ", m ) ;
}
```

And here's the output...

55 65 75 56 78 78 90