Welcome to C-programming Class

1 (a)

(a) What are the basic data types supported in C programming language?

Data Type	Description
int	Used to store integer values
float	Used to store single-precision decimal numbers
double	Used to store double-precision decimal numbers
char	Used to store a single character
void	Represents absence of value

1 (b)

Q1 (b) Describe the compilation process in C programming language

1. Preprocessing

Handles preprocessor directives like #include, #define. Removes comments.

Example:

#include<stdio.h>

1 (b)

Q1 (b) Describe the compilation process in C programming language

2. Compilation
Converts preprocessed code into assembly code. Checks syntax errors.

3. Assembly
Converts assembly code into machine code (object file).

1 (b)

Q1 (b) Describe the compilation process in C programming language

4. Linking

Links object file with library files.

Generates the executable file (.exe).

1 (b)

Q1 (b) Describe the compilation process in C programming language

Source Code → Preprocessor → Compiler → Assembler → Linker → Executable File.

1(c)

Difference between Entry Controlled Loop and Exit Controlled Loop

Entry Controlled Loop	Exit Controlled Loop
Condition checked before execution	Condition checked after execution
Loop may execute zero times	Loop executes at least once
Example: `for`, `while`	Example: `do-while`

1(c). While vs Do–While

While Loop	Do–While Loop
Entry-controlled	Exit-controlled
Condition checked first	Condition checked after execution
May execute 0 times	Executes at least once

syntax of do-while loop

```
do
{
    // statements
}
while(condition);
```

syntax of do-while loop

```
do
{
    // statements
}
while(condition);
```

Important Points:

do block runs first

Condition checked after execution

Semicolon (;) is mandatory after while(condition)

Simple c example

Print numbers from 1 to 5 using do–while loop.

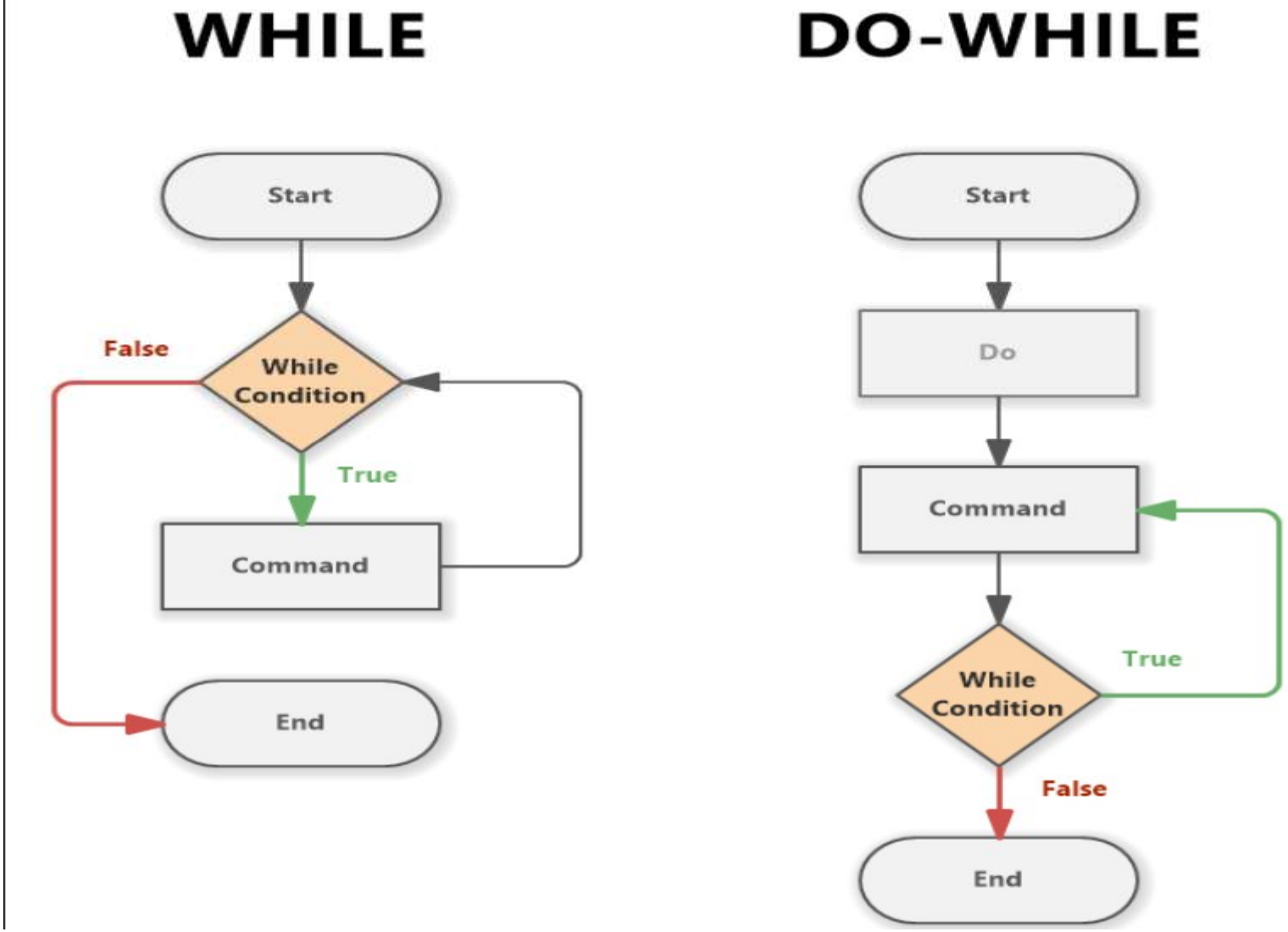
```
#include <stdio.h>
int main()
{
    int i = 1;
    do
    {
        printf("%d\n", i);
     }
    while(i <= 5);
    return 0;
}
```

Simple c example

Print numbers from 1 to 5 using do–while loop.

```
#include <stdio.h>
int main()
{
    int i = 1;
    do
    {
        printf("%d\n", i);
        i++;
    }
    while(i <= 5);
    return 0;
}
```

Flowchart


WHILE
Start
False
While Condition
True
Command
End
DO-WHILE
Start
Do
Command
While Condition
True
False
End

FOR LOOP

```
for(initialization; condition;
increment)
{
   // statements
}
```

output??

i=1 → True → Print 1
i=2 → True → Print 2
i=3 → True → Print 3
i=4 → True → Print 4
i=5 → True → Print 5
i=6 → False → Stop

```
for(int i=1; i<=5; i++)
{
   printf("%d", i);
}
```

COMMON ERRORS

- Missing increment
- Wrong condition
- Infinite loops
- Missing semicolon in do-while

1(d)

Q1 (d) Find the output of the following program

```
#include<stdio.h>
void main()
{
    int x=4,y,z;
    y=--x;
    z=x--;
    printf("%d,%d,%d\n",x,y,z);
}
```

1(d)

Q1 (d) Find the output of the following program

```
#include<stdio.h>
void main()
{
    int x=4,y,z;
    y=--x;
    z=x--;
    printf("%d,%d,%d\n",x,y,z);
}
```

1(d)

```
#include<stdio.h>      // Includes the standard input-output library for using printf()

void main()            // Main function where program execution starts
{
   int x=4,y,z;       // Declare three integer variables: x initialized to 4, y and z
uninitialized

y=--x;              // Pre-decrement: first decrease x by 1 (x becomes 3), then assign
value to y
                    // So y = 3

z=x--;              // Post-decrement: first assign value of x to z (z = 3), then decrease x
by 1
                    // Now x becomes 2

   printf("%d,%d,%d\n",x,y,z);   // Prints values of x, y and z respectively
                          // x = 2, y = 3, z = 3
}
```

1(d)- Dry-run:-

Step	Operation	Value of x	Value of y	Value of z
Initial	`int x=4`	4	-	-
`y = --x`	Pre-decrement	3	3	-
`z = x--`	Post-decrement	2	3	3

1(d)- Dry-run:-

Step	Operation	Value of x	Value of y	Value of z
Initial	`int x=4`	4	-	-
`y = --x`	Pre-decrement	3	3	-
`z = x--`	Post-decrement	2	3	3

key concept

Pre-decrement (--x) means value decreases before assignment

Post-decrement (x--) means value decreases after assignment
Initial

key concept

x=4
After y=--x
x=3
y=3
After z=x--
z=3
x=2

Final Values
x=2 , y=3 , z=3

Pre-processor directives

Q1 (e) What are preprocessor directives? Discuss #define with example

Preprocessor directives are commands processed before compilation. They start with the symbol #.

Examples:
#include
#define
#undef
#ifdef

Used to define symbolic constants or macros.

Pre-processor directives

Q1 (e) What are preprocessor directives? Discuss #define with example

Example:

```
#include<stdio.h>
#define PI 3.14

void main()
{
   float r = 5;
   float area = PI * r * r;
   printf("Area = %f", area);
}
```

Q2 (a) Program in C to swap two numbers using third variable

```
#include<stdio.h>
int main()
{
    int a, b, temp;
    printf("Enter two numbers: ");
    scanf("%d %d", &a, &b);
    temp = a;
    a = b;
    b = temp;
    printf("After Swapping:\n");
    printf("a = %d\n", a);
    printf("b = %d\n", b);
    return 0;
}
```

Q2 (b) Difference between break, continue and goto statement

Statement	Purpose
break	Terminates loop or switch immediately
continue	Skips current iteration and continues next iteration
goto	Transfers control to a labeled statement

BREAK STATEMENT

Definition

The break statement is used to terminate the loop or switch statement immediately.

Control moves to the next statement after the loop/switch.

SYNTAX: break;

```
#include <stdio.h>
int main()
{
    int i;
    for(i = 1; i <= 10; i++)
    {
        if(i == 5)
            break;
        printf("%d ", i);
    }
    return 0;
}
```

SYNTAX: break;

```
#include <stdio.h>
int main()
{
    int i;
    for(i = 1; i <= 10; i++)
    {
        if(i == 5)
            break;
        printf("%d ", i);
    }
    return 0;
}       output:- 1234
```

SYNTAX: break in switch

```
switch(choice)
{
    case 1:
        printf("Add");
        break;
    case 2:
        printf("Subtract");
        break;
}
```

Continue statement

The continue statement skips the current iteration and moves to the next iteration of the loop.It does NOT terminate the loop.

Continue statement

```
#include <stdio.h>
int main()
{
    int i;
    for(i = 1; i <= 5; i++)
    {
        if(i == 3)
            continue;
        printf("%d ", i);
    }
    return 0;
}
```

Continue statement

```
#include <stdio.h>
int main()
{
    int i;
    for(i = 1; i <= 5; i++)
    {
        if(i == 3)
            continue;
        printf("%d ", i);
    }
    return 0;
} output:- 1245
```

Difference between beak and continue

Feature	break	continue
Terminates loop	Yes	No
Skips iteration	No	Yes
Used in switch	Yes	No

GOTO STATEMENT

Definition

The goto statement transfers control to a labeled statement within the same function.

GOTO STATEMENT:-

```
goto label;
...
label:
   statement;
```

goto:-

```
#include <stdio.h>
int main()
{
    int i = 1;

start:
    printf("%d ", i);
    i++;

    if(i <= 5)
        goto start;

    return 0;
}
```

goto statement:-

```
#include <stdio.h>
int main()
{
    int i = 1;

start:
    printf("%d ", i);
    i++;

    if(i <= 5)
        goto start;

    return 0;
}                   output :- 1 2 3 4 5
```

Q3 (a) What is a function? Explain four categories of functions

A function is a block of code that performs a specific task and can be reused.

General Syntax

```
return_type function_name(parameters)
{
   // statements
}
```

Q3 (a) What is a function? Explain four categories of functions

Four categories of functions in C

Type
No arguments, No return value
Arguments, No return value
No arguments, Return value
Arguments, Return value

Q3 (a) What is a function? Explain four categories of functions

Four categories of functions in C

1.
No arguments, No return value

```
void display()
{
    printf("Hello");
}
```

Q3 (a) What is a function? Explain four categories of functions

2

Arguments, No return value

```
void add(int a, int b)
{
    printf("Sum = %d", a+b);
}
```

Q3 (a) What is a function? Explain four categories of functions

3

No arguments, Return value

```
int getNumber()
{
    int x=10;
    return x;
}
```

Q3 (a) What is a function? Explain four categories of functions

4

Arguments, Return value

```
int add(int a,int b)
{
    return a+b;
}
```

3 (b) C program to read 5 numbers in an array and display in reverse order

```
#include<stdio.h>
int main()
{
    int a[5], i;
    printf("Enter 5 numbers:\n");
    for(i=0;i<5;i++)
    scanf("%d",&a[i]);
    printf("Reverse Order:\n");
    for(i=4;i>=0;i--)
    printf("%d ",a[i]);
    return 0;
}
```

Q4 (a) Program to check whether a number is prime or not

```
int main()
{
   int n,i,flag=0;
   printf("Enter a number: ");
   scanf("%d",&n);
   for(i=2;i<=n/2;i++)
   {
      if(n%i==0)
      {
         flag=1;
         break;
      }
   }
   if(flag==0 && n>1)
   printf("Prime Number");
   else
   printf("Not Prime Number");
   return 0;
}
```

Q4 (a) Program to check whether a number is prime or not

```
#include<stdio.h>      // Includes standard input-output library for printf() and scanf()
int main()             // Program execution starts from main()
{
   int n,i,flag=0;     // Declare variables: n (input number), i (loop counter), flag=0 (assume number is
prime)
   printf("Enter a number: ");   // Display message asking user to enter a number
   scanf("%d",&n);               // User enters 7 → now n = 7
   if(n<=1)                      // Check if number is less than or equal to 1
   {                             // Prime numbers must be greater than 1
      printf("Not Prime");       // If n <= 1 then number is not prime
   }
   else                          // If n > 1 program enters else block
   {
      for(i=2;i<=n/2;i++)        // Loop starts from i=2 to n/2
                                 // For n=7 → n/2 = 3 → loop runs for i=2 and i=3
      {
         if(n%i==0)              // Check if n is divisible by i
                                 // Iteration 1: i=2 → 7 % 2 = 1 → not divisible
                                 // Iteration 2: i=3 → 7 % 3 = 1 → not divisible
         {
            flag=1;              // If divisible → set flag=1 (number is not prime)
            break;               // Exit loop immediately
         }
      }
      if(flag==0)                // After loop check flag value
                                 // flag is still 0 → no divisor found
      printf("Prime Number"); // Therefore 7 is a prime number
      else
      printf("Not Prime Number"); // This would execute if flag=1
   }    return 0;                 // End of program
```

4 (b) storage class

Storage classes define the scope, lifetime, and visibility of variables.

Storage Class	Description
auto	Default storage class
register	Stored in CPU register
static	Retains value between function calls
extern	Used for global variables

Array:- collection of similar data types

An array is a collection of similar data types stored in continuous memory locations.

Array:- collection of similar data types

Instead of writing: int m1 = 80, m2 = 75, m3 = 90

We use:

int marks[3] = {80, 75, 90}

Array

Concept of Array:- An array is a collection of similar data types stored in continuous memory locations.

Instead of writing: int m1 = 80, m2 = 75, m3 = 90
We use: int marks[3] = {80, 75, 90}

Memory Concept

Index:	0	1	2
Value:	80	75	90

Index always starts from 0.

1 D Array

One-Dimensional Array (1D Array)

Declaration

```
int arr[5];
```

Initialization

```
int arr[5] = {1, 2, 3, 4, 5};
```

Accessing Elements

```
printf("%d", arr[0]);
```

1 D Array C Example:-

```
#include <stdio.h>
int main() {
    int arr[5] = {10, 20, 30, 40, 50};
    int i;
    for(i = 0; i < 5; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

1 D Array C Apllications in AIML and ECE

AIML relevance: Used to store dataset features

ECE relevance: Used to store signal samples

2 D Array

Used for matrices (rows & columns).

2 D Array

Used for matrices (rows & columns).

Declaration
int matrix[2][3];

2 D Array

Used for matrices (rows & columns).

Declaration

```
int matrix[2][3];
```

Initialization

```
int matrix[2][3] = {
    {1, 2, 3},
    {4, 5, 6}
};
```

2 D Array

Used for matrices (rows & columns).

Declaration

```
int matrix[2][3];
```

Initialization

```
int matrix[2][3] = {
    {1, 2, 3},
    {4, 5, 6}
};
```

Accessing

```
printf("%d", matrix[1][2]);
```

2 D Array Example:- addition of an array

```
#include <stdio.h>
int main() {
    int a[2][2] = {{1,2},{3,4}};
    int b[2][2] = {{5,6},{7,8}};
    int c[2][2];
    int i,j;
    for(i=0;i<2;i++){
        for(j=0;j<2;j++){
            c[i][j] = a[i][j] + b[i][j];
        }
    }
    printf("Result Matrix:\n");
    for(i=0;i<2;i++){
        for(j=0;j<2;j++){
            printf("%d ", c[i][j]);
        }
        printf("\n");
    }
}
```

2 D Array Applications

AIML:
Used in neural network weight matrices

ECE:
Used in image processing

Multi-Dimensional Arrays

More than 2 dimensions

int arr[2][3][4];

Used in:

AI image datasets (RGB images)

3D signal processing

User-Defined Functions-definition

User Defined Functions:- Block of code written by programmer.

User-Defined Functions-Syntax

User Defined Functions:- Block of code written by programmer.

Syntax:-

```
return_type  function_name(parameters) {
    // body
}
```

User-Defined Functions- Example

Syntax

```
return_type  function_name(parameters) {
    // body
}
```

```
#include <stdio.h>
int add(int a, int b) {
    return a + b;
}
int main() {
    int result = add(5, 3);
    printf("Sum = %d", result);
    return 0;
}
```

Built-in Functions or user-defined functions

Provided by C library.

Examples:

```
sqrt(25);    // math.h
strlen("AI"); // string.h
```

Built-in Functions or user-defined functions

Provided by C library.

Examples:

```
sqrt(25);    // math.h
strlen("AI"); // string.h
```

Storage Classes

A storage class in C defines:

Scope → Where the variable can be accessed
Lifetime → How long the variable exists
Default initial value
Memory location

Storage Classes

Types of Storage Classes in C
auto
register
static
extern

Storage Classes

Storage Class	Scope	Lifetime
auto	Local	Inside block
static	Local	Entire program
extern	Global	Entire program
register	Local	CPU register

Storage Classes

AUTO Storage Class

✔ Default storage class for local variables

✔ Stored in main memory (RAM)

Scope:

Inside the block/function

Lifetime:

Until function execution ends

Default value: Garbage value

Storage Classes

Example

```
#include <stdio.h>
int main() {
    auto int x = 10;   // auto is optional
    printf("%d", x);
    return 0;
}
```

Storage Classes

Example

```
#include <stdio.h>
int main() {
    auto int x = 10;   // auto is optional
    printf("%d", x);
    return 0;
}
```

Note: Writing auto is optional because all local variables are auto by default.

Storage Classes

REGISTER Storage Class
Stored in CPU register (if available)
Used for fast access

Scope: Inside the block

Lifetime: Till function execution

Default value: Garbage value

Storage Classes

Example

```
#include <stdio.h>
int main() {
    register int i;
    for(i = 0; i < 5; i++) {
        printf("%d ", i);
    }
    return 0;
}
```

Storage Classes

Example

```
#include <stdio.h>
int main() {
    register int i;
    for(i = 0; i < 5; i++) {
        printf("%d ", i);
    }
    return 0;
}
```

Limitation:

You cannot use &i because register variables may not have memory address.

Storage Classes

Example: Without Static

```
#include <stdio.h>
void counter() {
    int count = 0;
    count++;
    printf("%d\n", count);
}
int main() {
    counter();
    counter();
    counter();
    return 0;
}
```

Storage Classes

Example: Without Static

```
#include <stdio.h>
void counter() {
    int count = 0;
    count++;
    printf("%d\n", count);
}
int main() {
    counter();
    counter();
    counter();
    return 0;
}
```

Output:

```
1
1
1
```

Storage Classes: C program as example

```
#include <stdio.h>
void counter() {
    static int count = 0;
    count++;
    printf("%d\n", count);
}
int main() {
    counter();
    counter();
    counter();
}
```

Storage Classes: C program as example

```
#include <stdio.h>
void counter() {
    static int count = 0;
    count++;
    printf("%d\n", count);
}
int main() {
    counter();
    counter();
    counter();
}
```

Output:

1

2

3

Storage Classes

STATIC Storage Class
Retains value between function calls
Stored in data segment
Scope:
Local to function (if declared inside function)

Lifetime: Entire program

Default value: 0

Call by Value

Copy of variable is passed.

```
void change(int x) {
    x = 100;
}

int main() {
    int a = 10;
    change(a);
    printf("%d", a);   // 10
}
```

Call by Reference (Using Pointers)

Original value changes.

```
void change(int *x) {
    *x = 100;
}
int main() {
    int a = 10;
    change(&a);
    printf("%d", a);   // 100
}
```

Passing Array to Function

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

Strings in C

Array of characters ending with \0

char name[] = "AIML";

Memory:

A I M L \0

Inputting Strings

```
char name[20];
scanf("%s", name);
```

Better method:

```
fgets(name, 20, stdin);
```

Character Library Functions

Include:

```
#include <ctype.h>
```

Examples:

```
toupper('a');  // 'A'
isdigit('5');  // 1
```

String Handling Functions

Include:

```
#include <string.h>
```

Function	Work
strlen()	Length
strcpy()	Copy
strcat()	Concatenate
strcmp()	Compare

Example:-

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

int main() {
    char a[20] = "AI";
    char b[] = "ML";

    strcat(a, b);
    printf("%s", a);    // AIML
}
```

Recursion:-

Recursion is a process in which a function calls itself until a stopping condition is reached. It consists of two parts: base condition and recursive call.

Factorial of a number.

```
int fact(int n)
{
    if(n==0)
        return 1;
    else
        return n * fact(n-1);
}
```

Recursion:-

```
#include <stdio.h>

int factorial(int n)
{
   if(n == 0)        // Base Case
      return 1;
   else
      return n * factorial(n - 1);   // Recursive Call
}

int main()
{
   int num = 5;
   int result;

   result = factorial(num);

   printf("Factorial = %d", result);

   return 0;
}
```

Recursion:-

Recursion is a process in which a function calls itself until a stopping condition is reached.

It consists of two parts: base condition and recursive call.

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

int main() {
    char a[20] = "AI";
    char b[] = "ML";
```

Questions

ARRAYS

Define an array. Why is it required in C?
What is the difference between 1D and 2D array?
What happens if array index goes out of bounds?
Write syntax of 2D array declaration.
What is a multidimensional array?

Questions

FUNCTIONS

What is a user-defined function?
Differentiate between call by value and call by reference.
Define recursion.
What is the purpose of static storage class?
What is the difference between local and global variables?

Questions

STRINGS

How are strings stored in C?
What is the role of '\0' in strings?
Write syntax of strlen() function.
Difference between gets() and fgets().
What is strcmp() used for?

Questions

FUNCTIONS

What is a user-defined function?
Differentiate between call by value and call by reference.
Define recursion.
What is the purpose of static storage class?
What is the difference between local and global variables?

God Bless you all

Thanks …